基于控制流的代码混淆技术研究*

蒋 华, 刘 勇, 王 鑫
(桂林电子科技大学 计算机科学与工程学院, 广西 桂林 541004)

摘 要: 为了提高基于垃圾代码的控制流混淆方法的优化效果,针对插入分支垃圾代码以及循环垃圾代码会引入大量额外开销的问题,从软件保护中代码混淆技术出发,对代码混淆技术的研究现状和原理、混淆算法攻击以及基于控制流混淆技术作了深入研究,提出一种基于 Java 代码控制混淆中插入垃圾代码的改进方法。新方法与基于垃圾代码的控制流混淆变换方法比较,结果表明,新方法增加了代码抵抗攻击者的静态分析的能力,增加了反编译以及逆向工程的难度,既达到了很好的防御逆向工程攻击的效果,又不会大量引入额外的系统开销。

关键词: 控制混淆; 代码; 窜改; 逆向工程; 分析
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2013)03-0897-03
doi:10.3969/j.issn.1001-3695.2013.03.066

Code confusion technology research based on control flow

JIANG Hua, LIU Yong, WANG Xin
(*School of Computer Science & Engineering*, *Guilin University of Electronic Technology*, *Guilin Guangxi* 541004, *China*)

Abstract: In order to improve optimization effect of control flow obfuscation algorithm based on garbage code. Aiming at the problem that it will cause a lot of extra costs when inserting branch garbage code and recycling garbage code, this paper used the code confusion technology from software protection to deeply discuss the theory and existing situation of code confusion technology, confusion algorithm attacking and the confusion technology based on control flow. It also gave an improved method that inserted garbage code in control confusion. New method compared with the control flow code based on junk confusion transform methods and the results show that new method increases the static analysis ability of the code that resists an attacker, increases the difficulty of de-compilation and reverse project, achieves a good effect of defending reverse project attacks, and reduces lots of system cost.

Key words: control confusion; code; tamper; reverse project; analysis

0 引言

目前混淆技术的研究中,新西兰 Auckland 大学的 Collberg 等人的成果最为显著。虽然现阶段混淆算法在复杂性度量和算法的实现上还需要作大量的工作,但由于其广阔的应用前景使越来越多的计算机科学工作者认识到了它的重要性,并作了一些研究。文献[1]从混淆技术的历史发展角度对现有的混淆技术理论、算法、攻击模式和评估进行了综述;文献[2]提出并实现了多分支语句的控制流迷惑技术和基于函数指针数组的迷惑技术;文献[3]提出了执行流重整混淆算法;文献[4]提出了基于抽象解释的代码迷惑有效性比较框架;文献[5]提出了构造分支垃圾代码和循环垃圾代码两种垃圾代码,用于增加程序反编译难度;文献[6]中提出基于混沌查找表的单向 hash 函数构造算法为模块插入数据。目前对代码混淆的研究工作主要是基于软件保护的,是对算法进行改进,但付出的代价都比较大,因为 Barak 等理论上证明了在终端运行且没有硬件辅助保护机制的代码混淆技术是不可能为代码提供彻底保护的[7]。

由于分支垃圾代码和循环垃圾代码会增加较大的执行开销,因此本文提出了标志符重命名并插入相似性垃圾代码的混淆技术,使用该技术不仅可以减小执行开销,而且还可以保证程序抵抗逆向工程的能力,从而达到了安全保护的目的。

1 相关理论

1.1 混淆的原理

代码混淆的目的是在不改变源程序功能的同时让程序的可读性尽量降低,使得反编译的代价超出攻击者通过反编译所获得的收益。其原理[8]是转换一个程序 p 到另一个程序 p',使得 p'对逆向工程师来说更为困难,同时 p'与 p 有以下相同的可观测行为:

a)若 p 不能结束或错误结束,则 p'也不能结束或错误结束。

b)否则 p'必须结束并产生与 p 相同的输出。

1.2 代码混淆的性能指标

对程序作混淆变换后的效果如何,通常从强度、耐受性、开销、隐蔽性四个方面来评估[9,10]。强度是指混淆变换究竟给原始程序增加了多大的复杂度;耐受性是指变换后的程序对使用自动去混淆工具进行攻击的抵抗度;开销是指经过混淆变换后的程序在执行时由变换所带来的额外执行时间和所需存储空间开销;隐蔽性是混淆后的程序与源程序的相似性。

收稿日期: 2012-07-18; **修回日期**: 2012-08-22 **基金项目**: 广西可信软件重点实验室基金资助项目(PF11041X)

作者简介: 蒋华(1963-),男,教授,主要研究方向为数据库系统与信息安全(jianghua@guet.edu.cn);刘勇(1987-),男,硕士,主要研究方向为计算机网络技术及网络信息安全;王鑫(1976-),男,副教授,主要研究方向为信息安全.

1.3 控制流混淆

Collberg 等人[9]将混淆技术分为布局混淆、数据流混淆、控制流混淆和预防性混淆四个种类。其中,对控制流混淆算法的研究相对较多,如 Cloakware 公司的平展控制流混淆算法[11]和不透明谓词混淆算法[10]。

控制流混淆的目的就是改变或复杂化程序的控制流,使程序更难以破译。通常代码是以逻辑方式划分成方法,并把相关代码放在一起。通过组合拆分不相关方法可以打破这种逻辑,使得攻击者无法理解代码之间的相互关系,从而达到保护软件源码的目的。在图 1、2 例子中,对于按次序执行的两个语句 A、B,增加一个控制条件来决定 B 的执行。通过这种方法加大逆向工程的难度,但是所有的干扰控制都不会影响 B 的执行。此处所加的控制条件必须是永远为真的,所有在其右边的分支始终不会执行。

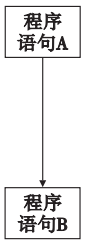

图1 顺序程序流图

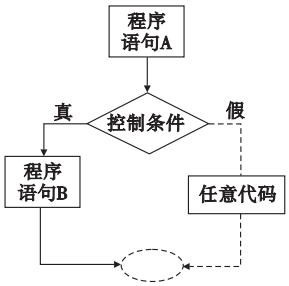

图2 插入任意代码混淆变换后的控制流图

1.4 对混淆算法的攻击

对混淆算法的攻击就是一个识别并移除的过程。从混淆后的程序中完全恢复出混淆前的源程序是攻击者的最终目的。目前主要攻击方法有模式识别、统计分析、数据流分析、动态分析、黑盒测试攻击等。动态分析技术是针对控制结构的算法,即在运行时观测各个模糊谓词的值。

对于控制结构混淆算法而言,变换后生成的程序中存在始终不执行的分支,通过动态分析就能找到有用的信息。黑盒测试攻击通过对程序作黑盒测试来了解类、函数和域的功能以获取信息。这种方法对大多数的混淆变换均能加以攻击。但是黑盒测试结果缺乏自动分析工具,需要大量的人力来完成分析工作。

2 代码混淆算法的实现

由于插入分支垃圾代码以及循环垃圾代码会给程序的执行带来很大的负担,因此本文提出一种基于控制流插入垃圾代码的改进算法。

2.1 垃圾代码

垃圾代码是指在 Java 的字节码数组中插入的代码,在 Java 程序运行时能被执行,但不改变程序的运行结果,并尽量降低系统开销,因此其具备如下特点:

a)无论 Java 程序中任何类变量、实例变量或局部变量是否被引用,不改变其值。

b)假设在控制流进入垃圾代码前栈的状态为 A,则无论其从何处离开垃圾代码,必须保持栈的状态也为 A。

c)垃圾代码可以在堆中创建对象,但不能改变堆中原来对象的状态。

d)垃圾代码不改变 Java 方法原来的控制流,即在插入垃圾代码前,原来的控制流是从 A 到 B,那么插入垃圾代码 C 之后,控制流从 A 到 C,执行完垃圾代码 C 之后,控制流必须转移到 B。

e)引入的代码与源代码差异较小[12]。如果一种变换所引入的代码和原始程序有较大的差异性,就会轻易地被攻击者识破,因此应尽力使用与原代码相近的语法结构等来加强隐蔽性。

由上可知,前面 a)~c)保障了原程序中指令的运行环境不会改变,d)保障了原来的控制流不会改变。因此,Java 方法的运行结果不会改变。

2.2 分支垃圾代码

其混淆变换过程如图 3 所示。

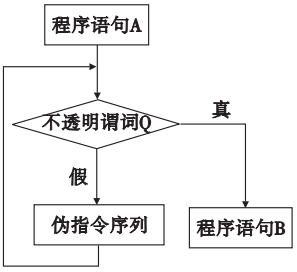

图3 插入垃圾代码后的控制流图

算法如下:输入给定程序的控制流图 G,输出程序的控制流图 G'。

a)考察控制流图 G 中如图 1 中的所有基本块(A、B);

b)while(G 中的基本块 B 不为空)

{

if(程序控制流离开 B 时,栈的状态完全相同,且栈顶为数值类型数据)在两者之间插入一个不透明谓词 Q

(a)添加 double 类型的局部变量,令其索引值为 index,在 B 前面加入指令序列:insert id,store index。

/* id 是一个基本块的唯一标志,insert 为任何浮点类型,数据压入栈中的指令,store 为任何把浮点类型数据存入局部变量中的指令 */

(b)引入一个类变量 t,令其常量池索引值为 index_pool。

(c)构造下列伪指令序列:open, putstatic index_pool, load index, lookupswitch。

/* lookupswitch 指令的跳转表按下列方式构造:id 与 id 对应的基本块中的某个后继,原来转向该后继的分支,现在转向 open、putstatic index_pool、load index、lookupswich */

(d)把构造的伪指令插入 B 之后,同时把(a)中所选基本块中被选中的分支指向 open、putstatic index_pool、load index、lookupswitch。

}

上述的构造实质是按照成为垃圾代码的条件来构思的。

2.3 标志符重命名

其基本原理是使用简单无意义的字符替换字节码文件中的类名、接口名、方法名以及字段名等。重命名方法的目的是使大型的 Java 程序更难理解或者更难被反编译,但不会引入程序的额外执行开销。首先,这些原始的标志符被一些无意义或者混乱的标志符代替,使得攻击者要想恢复这些原始的标志符变得不可能。因为与标志符相关的信息在重命名后完全消失。另外,这种方法也使得一个反编译器将字节码文件转换成 Java 源代码变得很困难甚至不可能,或者使得被反编译的源代码不能重新编译。由于标志符重命名算法在某种程度上能够增加反编译的难度,其实现也较为容易,所以本文提出在图 2

中控制条件处使用标志符重命名的方法来重命名控制条件中的变量或运算式。

2.4　Hash 函数的插入

除分支垃圾代码外,文献[5]中还构造了循环垃圾代码的混淆算法。由于循环次数决定着垃圾代码的执行次数,因此在循环中加入垃圾代码这种方法在实际应用中不可取。经过分支垃圾代码以及循环垃圾代码混淆变换的程序势必会增加一定的执行时间和所需存储空间的开销,甚至可能会因为耗尽内存空间而无法运行。所以需要对上述处理方法进行改进,限制插入垃圾代码的代码量以及这些垃圾代码的执行次数。

改进控制流混淆算法的思想是:在对不透明谓词(控制条件)进行标志符重命名处理的基础上,当操作块数过大时(大于某个值 M),则采用 hash 函数选取操作模块,进行混淆操作以限制混淆操作的次数,减小程序混淆后的时空开销。此处限制混淆操作次数包括两个 hash 函数插入,用来控制混淆变换次数。其中一个 hash 函数是控制如图 2 中所加任意代码的代码量,另一个 hash 函数是控制如图 3 中所添加垃圾代码的执行次数。Hash 函数插入如图 4 所示,其中 W-1 和 W-2 为程序图 1 的结构集合。

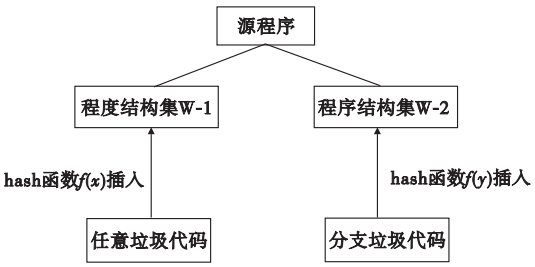

图4　不同的哈希函数插入垃圾代码示意图

3　算法性能比较分析

3.1　理论分析比较

1)隐蔽性

在设计垃圾代码特点 e)中,考虑到插入差异性较大的垃圾代码时,会很容易被攻击者静态分析所识破,相比文献[5]中所提出的方法更为严谨,使得被攻击者反编译出的代码更难以理解。因此,新混淆变换发放的隐蔽性比插入垃圾代码的混淆变换方法要强。

2)执行开销

在设计分支代码时,将源码中的部分分支模块采用如图 2 的混淆变换,因此并不是所有插入的垃圾代码都会执行,而且在循环模块中没有加入任何垃圾代码。然而文献[5]所提出的混淆方法中插入了垃圾代码,这些垃圾代码在程序运行时,会大量增加程序执行开销,更严重者甚至会导致程序因为资源不足而崩溃。

3)强度

设计的混淆方法在程序复杂度方面比文献[5]中所提方法低,在此层面上其强度稍弱,但在混淆变换的多样性方面,由于设计该混淆变换的目的在于实用性以及安全性,设计混淆变换的方法时更为多样化,这使得混淆变换的强度也大为增加。网络现有的一些反编译软件,其反编译的针对性较强,对经过多种混淆变换的程序,被反编译出来的代码更难以理解,且不容易对该程序逆向工程。因此整体强度方面本文所设计的混淆变换方法较强。

3.2　实验结果对比

通过基于控制流混淆算法执行的 CPU 时钟周期和成功率来验证该混淆算法解决实际问题的有效性和可行性。选取六份 Java 程序代码,分别将这六份程序代码编号为 1 ~ 6,其代码长度分别为 L1 ~ L6 。代码长度满足数学关系:L1 < L2 < L3 < L4 < L5 < L6。

对上述程序代码分别用以下三种方法处理:

a)不作任何处理。

b)分支垃圾代码混淆变换以及循环垃圾代码混淆变换。

c)首先对程序中的分支进行标志符重命名处理,然后再对该程序作如图 2 的控制混淆变换以及如图 3 的混淆变换,所插入的垃圾代码量与 b)中一样。

比较这三种处理方法防窜改攻击能力,实验结果如图 5 和 6 所示。

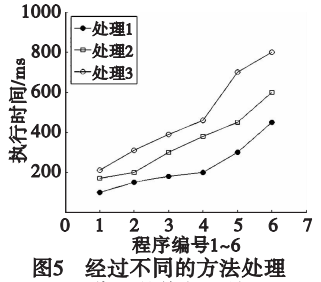

图5　经过不同的方法处理后代码的执行开销

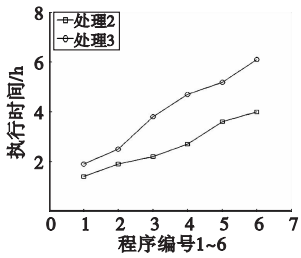

图6　防窜改攻击能力比较

程序执行开销是衡量一个混淆方法性能好坏的重要指标。由图 5 的结果可以看出,不论程序代码作何种代码混淆处理,混淆后程序的执行开销远比源程序大,而代码经过方法 c)处理比方法 b)所需的执行时间更少。方法 c)中所插入的任意垃圾代码,由于控制条件始终是为真的,所以其插入的垃圾代码是不会执行的,可以减小执行开销,但是增加的任意垃圾代码又会增加程序的复杂度。在防止窜改攻击方面,用方法 c)处理后的程序比方法 b)的防窜改能力要更强。实验结果如图 6 所示,在插入垃圾代码量相同的情况下,经过方法 c)处理后的代码能有效延长窜改攻击的时间,阻止攻击者对软件的静态分析以及逆向工程。因此,经过方法 c)处理后的程序具有更高的安全性。

4　结束语

本文首先分析了代码混淆技术的现状、原理以及混淆的性能指标,指出了控制流混淆的目的,然后介绍了控制流混淆原理,并提出实用性较高、安全性较好的控制流混淆算法。通过对提出的混淆算法性能分析比较,证明了所提出的混淆算法防窜改能力较强、实用性较好,因此能够很好地保护 Java 软件。在以后的工作中,希望针对混淆变换方法变换出的代码复杂度相对较低这方面加以改进,以便能够更有效地提高混淆变换方法的性能。

参考文献:

[1]　王建明,余志伟,王朝坤,等. Java 程序混淆技术综述[J]. 计算机学报, 2011,34(9):1578-1588.

(下转第 905 页)

对集中的情况下,前三种引擎把请求权重高的规则放在前面,从而提高按顺序匹配的效率。后两种提高程度缓和,这是由于 Enterprise XACML 和 MLOBEE 采用了策略索引优化,规则顺序对这两种引擎匹配效率的影响稍低。

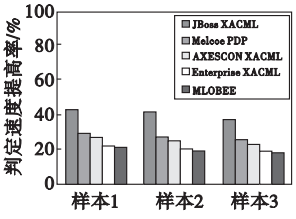

图2　规则优化性能分析a

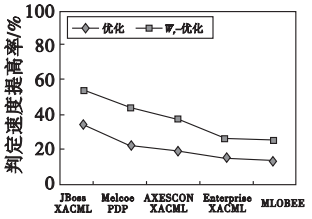

图3　规则优化性能分析b

4.2　优先级合并算法评估性能测试

在策略匹配模式方面,JBoss XACML 和 Melcoe PDP 都是以 XACML 作为引擎核心模块的系统,所以,图 4 中的实验选择 JBoss XACML 和 Melcoe PDP 这两种引擎与应用了本文提出的优先级合并算法的 Sun PDP 进行比较。实验数据采用上面的请求用例,测试不同规则数目对评估时的影响。从实验结果可看出,使用优先级合并算法的 Sun PDP 有明显优势。使用优先级合并算法的规则匹配时间比前两种耗时更短;另外随着规则数目的增大,使用优先级合并算法的 Sun PDP 评估时间变化较缓,这是由于本文提出的评估方法能更快地定位到相关规则,遍历过程沿着最有可能匹配成功的规则进行操作,所以稍微降低了规则数目对评估时间的影响。

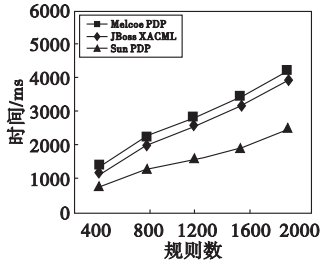

图 4　优先级合并算法评估性能比较

5　结束语

针对 XACML 判定评估问题的不足,分别从规则优化和评估方法上提出新的观点。对于请求相对集中(如多次重复)、较固定、但访问量大的情况,经过一定数量级的测试后,规则的请求权重将趋于一定值。采用规则按请求权重进行排序,可明显提高判定速率。另外,策略评估通过使用优先级合并算法及主体索引表可以快速定位到相关规则,缩短了请求的评估时间,进而提高了策略匹配效率。通过仿真测试分析,验证了该技术的有效性。今后的工作主要集中在对评估算法进行优化,使对多条断言的评估情况有更好的处理方法,进一步提升评估效率。

参考文献:

[1] OASIS. EXtensible access control markup language (XACML) version 3.0[S]. 2012.

[2] MARTIN E, XIE Tao, YU Ting. Defining and measuring policy coverage in testing access control policies[C]//Proc of the 8th International Conference on Information and Communications Security. Berlin:Springer-Verlag, 2006:139-158.

[3] GUELEY D P, RYAN M, SCHOBBENS P Y. Model-checking access control policies[C]//Proc of the 7th International Conference on Information Security. Berlin:Springer-Verlag,2004:219-230.

[4] FISLER K, KRISHNAMURTHI S, MEYEROVICH L A, *et al*. Verification and change-impact analysis of access-control policies[C]// Proc of the 27th International Conference on Software Engineering. New York: ACM Press,2005:196-205.

[5] HUGHES G, BULTAN T. Automated verification of XACML policies using a SAT solver[J]. Software Tools for Technology Transfer, 2008,10(6):503-520.

[6] JBoss XACML[EB/OL]. (2008). http://www.jboss.org/jbosssecurity/download/index.html.

[7] Melcoe PDP[EB/OL]. (2008). http://www.muradora.org/muradora/wiki/MelcoePDPDoc.

[8] Enterprise XACML[EB/OL]. (2008). http://code.google.com/p/enterprise-java-xacml/.

[9] 王雅哲,冯登国,张立武,等. 多层次优化技术的 XACML 策略评估引擎[J]. 软件学报, 2011,22(2):323-338.

[10] LIU A X, CHEN Feng, HWANG J H, *et al*. Designing fast and scalable XACML policy evaluation engines[J]. IEEE Trans on Computers,2011,60(12):1802-1817.

[11] KOLOVSKI V, HENDLER J , PARSIA B. Analyzing Web access control policies[C]//Proc of the 16th International Conference on World Wide Web. NewYork: ACM Press,2007:677-686.

[12] 王雅哲,冯登国. 一种 XACML 规则冲突及冗余分析方法[J]. 计算机学报,2009,32(3):516-530.

[13] XACML 2.0 conformance tests[EB/OL]. (2005). http://www.oasis-open.org/committees/download.php/14846/xacml2.0-ct-v.0.4.zip.

(上接第 899 页)

[2] 刁俊峰. 软件安全中的若干关键技术研究[D]. 北京:北京邮电大学,2007.

[3] 李长青,李晓勇,韩臻. 基于控制转换的软件保护[J]. 信息安全与通信保密,2006(10):146-149.

[4] 高鹰,陈意云. 基于抽象解释的代码迷惑有效性比较框架[J]. 计算机学报,2007,30(5):806-814.

[5] 杨乐,周强强,薛锦云. 基于垃圾代码的控制流混淆算法[J]. 计算机工程,2011,37(12):23-25.

[6] 邓绍江,李艳涛,张岱固,等. 基于混沌查找表的单向 hash 函数构造算法[J]. 计算机工程,2010,36(10):29-40.

[7] WANG Chen-xi, DAVIDON J, HILL J, *et al*. Protection of software-based survivability mechanisms[C]//Proc of International Conference on Dependable Systems and Networks. [S.l.]:IEEE Computer Society,2001:193-202.

[8] 宋亚奇. 基于代码混淆的软件保护技术研究[D]. 西安:西北大学,2005.

[9] COLLBERG C, THOMBORSON C, LOW D. A taxonomy of obfuscating transformations: a report[R][S.l.]: University of Auckland, 1997:148.

[10] COLLBERG C, THOMBORSON C, LOW D. Manufacturing cheap, resilient, and stealthy opaque constructs[C]//Proc of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York:ACM Press,1998:184-196.

[11] CHOW S, GU Y, JOHNSON H, *et al*. An approach to the obfuscation of control-flow of sequential computer programs[C]//Proc of the 4th International Conference on Information Security. London: Springer-Verlag, 2001:144-155.

[12] 张国宝. 基于 Java 的代码混淆研究[D]. 成都:电子科技大学,2008.