

基于抽象解释的代码迷惑有效性比较框架

高 鹰¹⁾ 陈意云^{1),2)}

¹⁾(中国科技大学计算机科学与技术系 合肥 230027)

²⁾(中国科学院软件研究所计算机科学实验室 北京 100080)

摘 要 代码迷惑是一种以增加理解难度为目的的程序变换技术,用来保护软件免遭逆向剖析.代码迷惑是否有效是代码迷惑研究中首要解决的问题.目前对有效性证明的研究大都是基于非语义的方式.文章将语义与有效性证明联系起来,建立了基于语义的代码迷惑有效性比较框架,该框架能够为迷惑算法在静态分析这样的限定环境下提供有效性证明,而且也能严格比较迷惑算法之间的有效性,最后使用实例描述比较框架如何应用到证明代码迷惑的有效性.

关键词 抽象解释;程序变换;程序分析;代码迷惑;压平算法

中图法分类号 TP311

A Comparable Code Obfuscation Framework Measuring Efficiency Based on Abstract Interpretation

GAO Ying¹⁾ CHEN Yi-Yun^{1),2)}

¹⁾(Department of Computer Science & Technology, University of Science & Technology of China, Hefei 230027)

²⁾(Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080)

Abstract Code obfuscation, which is an effective program transformation, can obscure the program understanding and thus protect the program from reverse engineering. There are a lot of applications about code obfuscation. This shows the efficiency of code obfuscation under some limited environments. So the proving of its efficiency is the prime problem of the research. But current research takes no account of the semantic information. This paper constructs a semantics-based comparable framework measuring obfuscation efficiency, which not only prove efficiency under the limited environment of static analysis, but also can establish the formal foundation for obfuscation efficiency comparison. The last part of the paper illustrates how the framework can be applied to measure the efficiency of code obfuscation with an instantiation.

Keywords abstract interpretation; program transformation; program analysis; code obfuscation; flattening algorithm

1 引 言

客户代码移动到主机执行时面临两类问题.一类是恶意客户问题,这时需要保护主机执行环境的

安全性不会受下载的客户代码的影响;另一类是恶意主机问题,这时需要保证客户代码不会被恶意主机窃取信息或篡改.代码迷惑是针对恶意主机问题而提出的一种保护客户代码的技术,它通过对代码进行程序变换,提高变换后代码的理解难度,来达到

保护客户代码的目的。

代码迷惑以前的研究主要集中在构造有效的代码迷惑算法. Collberg^[1] 中给出了有关这方面研究比较完整的综述, 引入了代码迷惑的定义, 代码迷惑是一种以增加理解难度为目的的程序变换技术. Wang^[2] 建立了针对恶意主机问题的代码安全体系, 该安全体系的主要部件基于代码迷惑技术构造, 而其中迷惑算法的核心思想是破坏程序的控制信息. Ogiso^[3] 推广了 Wang 的算法, 不仅破坏程序控制流信息, 还进一步地破坏程序过程间的调用信息. Douglas^[4] 针对 Java 语言的特征, 通过构造复杂数据结构来增加代码的理解难度. 关于构造代码迷惑算法的研究已经比较成熟.

但是, 对于构造的代码迷惑算法是否有效, 这些研究都没有提供严格证明, 代码迷惑算法的构造缺乏有效性证明的理论支持. 而另一方面, 许多研究已经从理论上证明了代码迷惑作为安全方法的局限性^[5-6], 即证明了不存在代码迷惑能够完全保证信息的安全性, 经过迷惑后的代码总还或多或少存在信息泄漏.

尽管在理论上代码迷惑并不能保证高机密信息的安全, 但代码迷惑仍是代码安全问题中一种有效的安全技术, 其原因是在很多场合下它能提供安全性, 因此代码迷惑的研究一直活跃. 一些具有代表性的代码迷惑应用如下:

恶意移动代理. 移动代理在主机之间移动时, 代码和执行的中间结果可能会被主机恶意获取篡改. SPMA^[7] 的研究表明, 代码迷惑技术能够保证代理在其移动存活期内被攻击的难度增加, 从而达到保护移动代理的安全. 对于恶意攻击者来说, 恶意移动代理是一种时间受限环境, 因为代理驻留在主机上的时间是有限的.

恶意逆向工程. 越来越多的代码使用容易被反编译的中间代码发布, 使得软件开发者需要更多地考虑其竞争者可能会反编译发布的代码, 进而获取软件的设计以及其中的重要算法. 虽然代码迷惑技术无法为代码提供完全保护. 但是, 代码迷惑能够使攻击者发现: 复用反迷惑后得到的代码要比其重写等效代码更加困难.

这些都说明代码迷惑能够提供限定环境下的安全性, 如上述的时间受限环境以及不可复用环境等. 限定环境是针对攻击者所处的攻击场景受到某些限制的情况, 如在时间受限环境下, 攻击者必须在代理存活期内篡改程序, 攻击者的攻击时间是受限的.

针对目前代码迷惑研究中缺乏基于语义的有效性证明以及缺乏限定环境下有效性证明的问题, 本文以代码迷惑引起的语义信息变化来刻画有效性, 提出了与语言无关的代码迷惑有效性比较框架, 能够为迷惑算法在静态分析这样的限定环境下提供严格的有效性证明, 也能够严格比较不同迷惑算法之间的有效性. 静态分析作为限定环境是指, 攻击者使用静态分析作为攻击手段的攻击场景.

本文第 2 节概述代码迷惑有效性比较框架组成部分; 第 3 节采用抽象解释理论形式化有效性比较框架; 第 4 节结合具体的迷惑算法描述如何实例化有效性比较框架; 最后给出相关工作比较和结论.

2 问题的提出

建立代码迷惑有效性比较框架分两个部分进行: 形式化代码迷惑空间; 形式地定义代码迷惑有效性度量.

第一步, 形式化迷惑算法组成的代码迷惑空间.

关于代码迷惑, Collberg^[1] 给出了非形式的定义.

定义 1. 代码迷惑. 程序变换 τ_{ob} 是代码迷惑是指: (1) 变换 τ_{ob} 保持程序可观察语义的等价性; (2) 经过变换 τ_{ob} 使得程序某些属性的理解难度增加.

根据上述定义, 代码迷惑包含了两个性质, 与程序变换一样保证程序可观察语义等价以及使得属性的理解难度增加. 形式化代码迷惑定义, 也需要从刻画这两个性质进行.

代码迷惑是一类特殊的程序变换, 因此与程序变换同样需要满足正确性性质. 程序变换正确性是指变换前和变换后程序具有可观察语义等价性. 由文献[8]可知, 可观察语义是对标准语义的抽象, 程序变换正确性等价于要求变换前后程序的标准语义在某种层次的抽象下相等. 这为定义代码迷惑正确性提供了理论基础. 属性是指从程序中提取的信息, 可以使用上界闭包来刻画^[9], 属性组成的属性空间可定义偏序结构, 属性的理解难度增加是通过属性空间上的偏序关系来定义.

第二步, 给出代码迷惑的有效性度量. 代码迷惑的有效性证明是基于定义的有效性度量. 我们采用语义信息的变化来刻画有效性, 通过迷惑前后的语义信息来度量代码迷惑的有效性. 语义信息通常是不可计算的, 静态分析是对程序语义信息的保守近似, 静态分析的结果能够可计算地反映语义信息的变化. 而且静态分析具有动态分析所不具备的可靠

性,能够保守地反映程序的性质.许多研究者^[1-4,7,10]都采用静态分析建立攻击模型.因此,除不可计算语义信息外,也需要采用静态分析得到的语义信息来度量有效性.

采用静态分析得到的语义信息来刻画有效性,是指对迷惑前和迷惑后程序进行分析,如果得到的分析集合变得更加庞大,则称迷惑算法是有效的.因为迷惑使得静态分析工具只能得到平凡的结果.

使用流图语言给出一个简单的例子,来解释基于静态分析结果的有效性定义:关于流图语言的定义见第4节

程序 OP 为: $L_1: x := ? \rightarrow L_2;$

$L_2: y := 1 \rightarrow L_3; L_3: y := y * x \rightarrow L_{\text{exit}};$

在程序中插入一段死代码, L_3 处的分支条件 false 表示 L_5 是会被执行到的,得到变换后的程序 OP' 为

$L_1: x := ? \rightarrow L_2; L_2: y := 1 \rightarrow L_3;$

$L_3: \text{true} \rightarrow L_4; L_4: y := y * x \rightarrow L_{\text{exit}};$

$L_3: \text{false} \rightarrow L_5; L_5: x := 1 \rightarrow L_{\text{exit}}.$

到达定值分析的目的是分析程序中的赋值信息^[11],我们将看到死代码插入会导致分析结果精度的下降.

针对程序点 L_{exit} 入口处,来分析比较变换前程序 OP 和变换后程序 OP' 的分析结果:

对于程序 OP,分析得到的集合为

$$L_{\text{exit}} \{ (x, L_1), (y, L_2) \},$$

其中, $L_{\text{exit}} \{ (x, L_1) \}$ 表示在程序点 L_{exit} 的入口处存在着对 x 的赋值,括号里的第二个元素表示赋值发生的程序点.

对于变换后程序 OP', 分析得到的集合增大为 $L_{\text{exit}} \{ (x, L_1), (x, L_4), (y, L_3) \}$, 是变换前在该程序点分析集合的超集. 而且进一步计算整个程序的定值集合就能发现,变换后程序的分析集合也是变换前程序的分析集合的超集. 所以,死代码变换是一种代码迷惑,死代码变换对于到达定值分析来说是有效的.

3 建立代码迷惑有效性比较框架

首先简要介绍抽象解释^[9]. 经典抽象解释是一种针对计算机系统语义模型的近似理论. 抽象解释为不可计算语义建立了安全可靠的近似语义,通过可计算的近似语义来达到描述不可计算的语义的目的. 主要思想是对给定的程序设计语言赋予具体和

抽象两种语义,将不可计算语义定义成具体语义,将可计算的语义定义成抽象语义,然后建立二者间的正确性联系,通过对可计算的抽象语义的求解来达到保守地计算不可计算语义的目的. 后面我们将会给出不可计算语义和可计算语义的例子. 下面给出抽象解释的主要组成部分,抽象解释存在许多等价描述,本文使用与程序分析联系比较紧密的伽罗瓦联系来描述抽象解释框架.

定义 2. 伽罗瓦联系. 完全偏序集 $\langle \mathcal{L}, \sqsubseteq \rangle$ 与完全偏序集 $\langle \mathcal{L}^\#, \sqsubseteq^\# \rangle$ 满足伽罗瓦联系 (Galois Connection), 是指存在抽象函数 $\alpha: \mathcal{L} \rightarrow \mathcal{L}^\#$ 以及具体函数 $\gamma: \mathcal{L}^\# \rightarrow \mathcal{L}$, $\forall X \in \mathcal{L}, \forall X^\# \in \mathcal{L}^\#$, 满足关系:

$$\alpha(X) \sqsubseteq^\# X^\# \Leftrightarrow X \sqsubseteq \gamma(X^\#).$$

伽罗瓦联系也记为

$$\langle \mathcal{L}, \sqsubseteq \rangle \xleftrightarrow[\gamma]{\alpha} \langle \mathcal{L}^\#, \sqsubseteq^\# \rangle.$$

对程序语言 \mathbb{L} 中的程序 \mathbb{P} , 给出组成抽象解释框架两种语义的定义.

定义 3. 具体语义. 具体语义论域是由偏序集合 $\langle \mathcal{D}, \sqsubseteq^c \rangle$ 构成, 其中 \sqsubseteq^c 是定义在集合 \mathcal{D} 上的偏序, $S \in \mathbb{P} \rightarrow \mathcal{D}$, 是基于语言 \mathbb{L} 语法构造的到域 \mathcal{D} 上的指称, 称为具体域 \mathcal{D} 上的具体语义函数, 具体语义函数就构成了程序的具体语义.

定义 4. 抽象语义. 抽象语义论域是由偏序集合 $\langle \mathcal{A}, \sqsubseteq^a \rangle$ 构成, 其中 \sqsubseteq^a 是定义在集合 \mathcal{A} 上的偏序, $S^a \in \mathbb{P} \rightarrow \mathcal{A}$ 是基于语言 \mathbb{L} 语法构造的到域 \mathcal{A} 上的指称, 称为抽象域 \mathcal{A} 上的抽象语义函数, 抽象语义函数就构成了程序的抽象语义.

通常, 为保证求解终止性, 具体域和抽象域需要具有比偏序集更强的性质. 下面提到的语义论域都是指完全偏序或是完全格. 通过寻找伽罗瓦联系, 建立具体域和抽象域间的联系:

$$\langle \mathcal{D}, \sqsubseteq^c \rangle \xleftrightarrow[\gamma]{\alpha} \langle \mathcal{A}, \sqsubseteq^a \rangle \quad (1)$$

在建立了论域间的伽罗瓦联系后, 抽象语义通过式 $S^a = \alpha \circ S$ 建立了对具体语义的抽象.

这样就建立了抽象解释框架, 抽象解释框架只考虑语义之间的正确性联系, 而忽略语义定义的细节, 因此具体语义和抽象语义是通用的概念, 需要根据不同的应用对二者进行实例化, 下面给出几组需要用到的抽象解释框架下的实例.

定义 5. 标准语义. 它是指定义语言的动态语义, 通常是指对语言的标准指称解释, 其中标准语义论域由偏序集 $\langle \mathcal{D}_s, \sqsubseteq_s \rangle$ 组成, $\mathcal{S}_s: \mathbb{P} \rightarrow \mathcal{D}_s$ 为标准语义函数.

定义 6. 可观察语义. 它是指程序中针对某些

属性在语义域上的感兴趣的取值,其中 \mathcal{D}_o 是可观察语义论域,一般与标准语义域 $\langle \mathcal{D}_s, \sqsubseteq_s \rangle$ 相同。 $\mathcal{O}: \mathbb{P} \rightarrow \mathcal{D}_s$ 为可观察语义函数。

标准语义提供了程序动态运行行为的精确定义,是其它一切语义模型建立抽象的起点,所有其它的语义都是对标准语义的抽象.而可观察语义正是对标准语义某个方面的抽象.若建立了标准语义域到可观察语义域之间的伽罗瓦联系:

$$\langle \mathcal{D}_s, \sqsubseteq_s \rangle \xleftrightarrow[\gamma_o]{\alpha_o} \langle \mathcal{D}_o, \sqsubseteq_o \rangle \quad (2)$$

又因为抽象语义可以通过式子 $S^a = \alpha \circ S$ 建立了对具体语义的抽象,在建立了论域间的伽罗瓦联系后,可观察语义就可由 $\mathcal{O} = \alpha_o \circ S_s$ 定义。

抽象解释框架应用到静态程序分析时,具体语义和抽象语义分别实例化成收集语义和分析语义。

定义 7. 收集语义. 它是指可到达程序动态运行行为的并集. 收集语义论域 $\langle \mathcal{D}_{col}, \sqsubseteq_{col} \rangle$, $\mathcal{D}_{col} \triangleq \wp(\mathcal{D}_s)$, 由标准语义论域的幂集组成, $S_{col}: \mathbb{P} \rightarrow \mathcal{D}_{col}$ 为收集语义函数。

定义 8. 分析语义. 它是指静态分析时基于收集语义得到的保守语义信息. 分析语义论域由偏序集 $\langle \mathcal{A}_\varphi, \sqsubseteq_\varphi \rangle$ 组成, $S_\varphi: \mathbb{P} \rightarrow \mathcal{A}_\varphi$ 为分析语义函数。

对于静态分析算法 φ , 可以建立收集语义域 $\langle \mathcal{D}_{col}, \sqsubseteq_{col} \rangle$ 到分析语义域 $\langle \mathcal{A}_\varphi, \sqsubseteq_\varphi \rangle$ 间的伽罗瓦联系:

$$\langle \mathcal{D}_{col}, \sqsubseteq_{col} \rangle \xleftrightarrow[\gamma_\varphi]{\alpha_\varphi} \langle \mathcal{A}_\varphi, \sqsubseteq_\varphi \rangle \quad (3)$$

同样,根据论域间建立的伽罗瓦联系,分析语义可由 $S_\varphi = \alpha_\varphi \circ S_{col}$ 定义。

对于例子中提到的到达定值分析,我们来看基于抽象解释框架是如何定义程序分析算法的. 首先是实例化具体语义得到收集语义,通常是到达程序点的所有状态的并. 对于程序 OP, 进入程序点 L_3 的状态为 $\{x := ?, y := ?, z := 1\}$, 这里的状态只包含了环境中值的映射, 因为 x 取值的不确定性, 使得程序点 L_3 处 y 值是不可解的, 因此这里的收集语义是不可计算的. 然后是实例化抽象语义得到分析语义, 是通过定义到达定值分析的抽象函数 $\alpha_\varphi: \mathcal{D}_{col} \rightarrow \mathcal{A}_\varphi$, 由收集语义构造得到. 限于篇幅, 这里不再给出 α_φ 的形式定义. 对于程序 OP 进入程序点 L_3 的分析语义是: $\{(x, ?), (y, L_1), (z, L_2)\}$, 表示所有到达程序点 L_3 的定值信息. 分析语义中关心的是定值信息, 程序点 L_3 处 y 的定值信息是可解的, 因此 x 定值信息的分析是可计算的。

本节主要内容是基于抽象解释框架, 建立代码迷惑有效性比较框架. 3.1 节给出属性组成的属性

空间的定义. 属性是形式化代码迷惑定义所需的, 只有定义了属性才能定义程序理解难度的增加; 3.2 节在给出的属性定义的基础上, 形式化代码迷惑组成的迷惑空间; 3.3 节基于程序分析框架, 建立有效性度量的标准, 为比较代码迷惑的有效性提供语义上的度量。

3.1 建立属性空间

程序在具体域上的取值称为具体属性, 在抽象域上的取值称为抽象属性, 通常具体属性都是不可计算的, 需要使用对应的可计算抽象属性来描述具体属性. 具体属性和抽象属性间的联系称为属性关系。

首先给出定义属性空间所需的上界闭包的定义。

定义 9. 上界闭包. 对于偏序集 $\langle \mathcal{P}, \sqsubseteq \rangle$, 算子 $\rho: \mathcal{P} \rightarrow \mathcal{P}$ 是上界闭包算子, 是指具备以下性质:

- (1) 单调的 (monotone). $\forall P, Q \in \mathcal{P}, P \sqsubseteq Q, \rho(P) \sqsubseteq \rho(Q)$;
- (2) 幂等的 (idempotent). $\rho \circ \rho = \rho$;
- (3) 外延的 (extensive). $\forall P \in \mathcal{P}, \rho(P) \sqsupseteq P$.

那么, $\rho(\mathcal{P})$ 就称为是偏序集 \mathcal{P} 关于闭包算子 ρ 的一个上界闭包. $uco(\mathcal{P})$ 表示偏序集 \mathcal{P} 上的全部上界闭包算子集合. 需要注意的是, 许多研究并不区分上界闭包算子和上界闭包的使用. 本文需要区分二者作为属性关系和属性的定义, 所以, 又引入 $UCO(\mathcal{P})$ 表示偏序集 \mathcal{P} 上的全部上界闭包集合。

抽象域可以使用上界闭包等价地刻画^[9]. 本文的抽象解释框架是使用伽罗瓦联系定义, 因此下面给出基于伽罗瓦联系的上界闭包算子定义. 由伽罗瓦联系(1), 可定义 $\rho_A = \gamma \circ \alpha$, $\rho_A \in \mathcal{D} \rightarrow \mathcal{D}$, 得到的 ρ_A 就是与抽象域 \mathcal{A} 相关的上界闭包算子. 对于抽象域 $\langle \mathcal{A}, \sqsubseteq^a \rangle$, 根据定义的 ρ_A , 存在关系 $\rho_A(\mathcal{D}) \cong \mathcal{A}$, 即 $\rho_A(\mathcal{D})$ 与 \mathcal{A} 具有相同的逻辑含义, 具体域的上界闭包 $\rho_A(\mathcal{D})$ 是抽象域 \mathcal{A} 的同构刻画。

使用上界闭包表示抽象域的好处是, 在推导抽象域上的属性时, 无需得到抽象域上的对象, 因为对于抽象域 $\langle \mathcal{A}, \sqsubseteq^a \rangle$, $\langle \rho_A(\mathcal{D}), \sqsubseteq^c \rangle$ 能同构地反映它的元素, 而 $\rho_A(\mathcal{D})$ 的构造与抽象域 \mathcal{A} 的定义是无关的。

程序的属性空间是由全体具体属性和抽象属性组成. 抽象域可由上界闭包等价刻画, 具体域 \mathcal{D} 上所有抽象域组成的偏序集 $\mathbb{L}_c(\mathcal{D}) = \langle UCO(\mathcal{D}), \sqsubseteq^c \rangle$. 由上界闭包算子的外延性可知: $\mathcal{D} \sqsubseteq UCO(\mathcal{D})$, 即具体域 \mathcal{D} 上的元素也属于 $UCO(\mathcal{D})$, 因此, 偏序集 \mathbb{L}_c 就组成了具体域 \mathcal{D} 的属性空间. $UCO(\mathcal{D})$ 上的偏序关系 \sqsubseteq^c 可以比较属性之间的语义的精度。

属性关系表示了具体属性和抽象属性的联系,

可以由上界闭包算子来定义,对于具体域 \mathcal{D} ,属性关系组成偏序集 $\mathcal{L}_c(\mathcal{D}) = \langle uco(\mathcal{D}), \sqsubseteq^r \rangle$, \sqsubseteq^r 表示算子之间的偏序,即 $\forall P \in \mathcal{D}, \rho \sqsubseteq^r \rho' \Rightarrow \rho(P) \sqsubseteq \rho'(P)$. 当 \mathcal{D} 为完全格时,能得到 \mathcal{L}_c 也为完全格. 因此,抽象解释格 \mathcal{L}_c 就组成了具体域 \mathcal{D} 的属性关系空间.

3.2 模型化迷惑空间

为形式地定义代码迷惑组成的迷惑空间,根据代码迷惑非形式定义,需要刻画两个性质:与程序变换一样需要保持程序可观察语义等价性以及属性的理解难度增加.

首先,代码迷惑是一种特殊的程序变换,因此需要具有与程序变换相同的可靠性和正确性.

(1) 根据变换规则是基于语法的或是基于语义的,程序变换可以分成两类:语法变换和语义变换. 这里 $\tau_{ob}: \mathbb{P} \rightarrow \mathbb{P}$ 来表示语法变换,它对于输入程序 $P \in \mathbb{P}$ 得到变换后程序 $\tau_{ob}[P]$; $t_{ob}: \mathcal{D} \rightarrow \mathcal{D}$ 表示语义变换,对于输入程序 P 的具体语义 $S[P]$,变换后得到语义 $t_{ob}[S[P]]$. 抽象解释是基于语义的形式化框架,使用语义来给出程序变换的规范,为证明语法变换 τ_{ob} 对程序的变化满足规范,需要建立与满足规范定义的语义变换 t_{ob} 之间的正确性联系. 为此,语法变换需要满足性质: $S[\tau_{ob}[P]] \sqsubseteq t_{ob}[S[P]]$. 在证明程序变换的可靠性时,这是需要证明满足的性质. 本文是讨论程序变换的语义信息的变化行为,并不涉及到语法变换需要满足一定语义规范的性质,因此,代码迷惑有效性比较框架不需要证明语法变换的可靠性性质.

(2) 程序变换的正确性是要求程序变换在可观察抽象 \mathcal{O} 下具有等价性. 由图 1,对于语法变换 τ_{ob} ,变换后的程序 $\tau_{ob}[P]$ 的具体语义为 $S[\tau_{ob}[P]]$;对于语义变换 t_{ob} ,基于程序 P 的具体语义 $S[P]$,经过语义变换后得到具体语义 $t_{ob}[S[P]]$;加上程序 P 的具体语义 $S[P]$,这三者之间需要满足可观察语义的等价性.

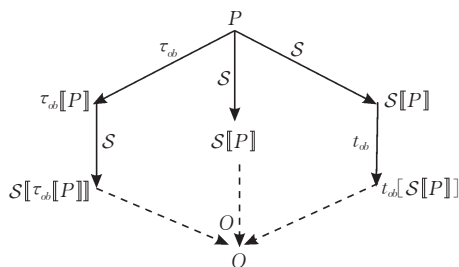


图 1 迷惑算法的正确性关系

即代码迷惑需要满足以下正确性定理:给定程

序 $P \in \mathbb{P}$, 那么等式

$$\mathcal{O}(S[P]) = \mathcal{O}(t_{ob}[S[P]]) = \mathcal{O}(S[\tau_{ob}[P]])$$

必须要成立,这就是代码迷惑的可观察语义等价性定理.

根据属性空间的定义,具体域 \mathcal{D} 的属性空间由偏序集 $\mathcal{L}_c(\mathcal{D})$ 组成,偏序关系意味着信息的丢失,我们使用 $\mathcal{L}_c(\mathcal{D})$ 上的偏序关系来反映属性的理解难度的增加,因此,可以得到代码迷惑形式的定义.

定义 10. 代码迷惑. 程序变换 τ_{ob} 是代码迷惑是指 τ_{ob} 使得某种属性关系 $p \in uco(\mathcal{D})$ 满足偏序关系:

$$p(S[P]) \sqsubseteq^c p(S[\tau_{ob}[P]]),$$

其中 \sqsubseteq^c 是具体域 $\langle \mathcal{D}, \sqsubseteq^c \rangle$ 上的偏序关系.

该定义反映了代码迷惑需要满足的两个性质:可观察语义等价和存在属性理解难度增加.

3.3 模型化迷惑有效性

为了比较代码迷惑的有效性,需要选择比较的度量. 本文主要采用以静态程序分析结果作为评价的基准,原因是:第一,基于语义角度的代码迷惑有效性评估,能够比基于语法的度量更加反映程序的本质,而且更加接近于逆向工程者的角度;第二,具体属性通常是不可计算的,需要得到具体属性的可靠保守解答,当然,不完备性使得抽象属性不能精确地回答具体域上的问题. 基于抽象解释框架下的程序分析定义,下面以命题的形式给出使用语义信息作为有效性比较的度量标准.

一般地,迷惑的有效性比较是基于可计算的语义上的.

命题 1(迷惑的有效性). $\forall P \in \mathbb{P}, \exists \varphi \in uco(\mathcal{D})$, 如果关系 $S_\varphi[P] \sqsubseteq_\varphi S_\varphi[\tau_{ob}[P]]$ 成立,则称变换 τ_{ob} 针对分析算法 φ 是有效的. 上界闭包算子 φ 是伽罗瓦联系的等效描述,当实例化为对程序分析框架时, φ 指代具体的分析算法.

抽象解释框架下,程序的收集语义和分析语义分别是具体语义和抽象语义的实例化,迷惑的有效性比较是基于收集语义或是分析语义,而有效性度量则是由论域上的偏序关系来确定. 这里的有效性度量与代码迷惑的定义是一致的,当然,为了使得有效性比较框架更加具有描述力,在此基础上可以使用与定义不一致的更加复杂的有效性度量定义.

传递定理. $\forall P, Q \in \mathbb{P}, S[P] \sqsubseteq^c S[Q] \Rightarrow S^a[P] \sqsubseteq^a S^a[Q]$.

证明. 由伽罗瓦联系(1), S 到 S^a 间可以建立单调的映射关系 $S^a = \alpha \circ S$, 由 α 的单调性且 $S[P] \sqsubseteq^c$

$S[Q]$, 可以保证结论成立.

传递定理也可以表述为抽象 α 是保偏序的.

推论. $\forall P, Q \in \mathbb{P}, S_{col}[P] \subseteq S_{col}[Q] \Rightarrow S_{\varphi}[P] \subseteq S_{\varphi}[Q]$.

传递定律及其推论的含义在于: 抽象解释将程序分析看成是语义的近似, 分析算法即是建立对语义的抽象; 而且程序分析算法之间可以建立相互之间的抽象. 所以可以根据传递定理, 只要对于某一种抽象的程序分析算法满足偏序, 那么这样的偏序关系是可以传递到其它由该抽象构造的其它程序分析算法上的.

因此, 如果在收集语义上可以比较迷惑有效性, 能得到更强的结论, 因为此时还不涉及到分析语义的实例化, 迷惑算法在收集语义上的有效性, 对于所有分析算法来说都是有效的.

所以, 迷惑的有效性比较是能够基于不可计算的语义上的.

命题 2 (迷惑的有效性). $\forall P \in \mathbb{P}$, 如果关系 $S_{col}[P] \subseteq_{col} S_{col}[\tau_{ob}[P]]$ 成立, 则称变换 τ_{ob} 针对任意分析算法都是有效的.

命题 1 和命题 2 为比较迷惑前后的程序的语义提供了理论基础. 命题 2 要求的是对通常是不可计算的收集语义上的比较, 通常无法直接计算得到二者间的关系, 但是可以得到比较强的结论, 是通过语义信息的变化来定义有效性的, 与具体的分析算法无关; 而命题 1 是建立在分析语义基础上的, 是基于分析得到的语义信息来定义有效性, 可以直接计算得到二者间的关系, 不过有效性只是适用于这个具体的分析算法.

4 实例化有效性比较框架

本节主要内容是根据上节提出的代码迷惑有效性比较框架, 针对一个具体的迷惑算法, 使用例子说明有效性比较框架证明迷惑算法有效性的过程. 为证明迷惑算法的有效性, 首先需要证明所比较的变换算法是属于迷惑空间的, 然后需要证明算法的有效性满足有效性命题. 为此, 需要实例化的部分有实例语言和迷惑算法. 通过定义实例语言, 就能够得到标准语义. 由抽象解释框架理论, 实例化框架所需要的其它语义, 包括可观察语义、收集语义、分析语义, 都能通过对标准语义的定义抽象得到. 在实例化这些部分后, 就可以严格证明保证代码迷惑正确性的可观察语义等价性定理以及说明代码迷惑有效性的

代码迷惑有效性定理.

4.1 定义实例语言

首先在流图语言^[8] (如图 2) 的基础上定义离散转移系统 $\mathbb{T} = \langle \Sigma, \Sigma_i, \iota \rangle$ 定义如下:

Σ 表示状态集合, $\Sigma \subseteq \mathfrak{S} \times \mathbb{C}$, 其中 \mathfrak{S} 是环境集合, 环境是变量到值的映射, \mathbb{C} 是命令集合. 状态 $s = \langle \rho, C \rangle$ 中, ρ 表示下一条命令 C 的执行环境.

$\Sigma_i \subseteq \Sigma$ 表示程序的初始状态.

程序	$P: \mathbb{P}$	$P \triangleq \varphi(C)$
命令	$C: \mathbb{C}$	$C ::= L_1; A \rightarrow L_2;$
程序动作	$A: \mathbb{A}$	$A ::= V := E \mid V := ? \mid skip \mid B$
算术表达式	$E: \mathbb{E}$	$E ::= n \mid x \mid E_1 op_a E_2$
布尔表达式	$B: \mathbb{B}$	$B ::= true \mid false \mid B_1 op_b B_2 \mid \neg E_2 \mid B_1 op_c B_2$
变量	$V: \mathbb{V}$	标签 $L: \mathbb{L}$ 整数 $n: \mathbb{Z}$

图 2 流图语言的语法 (其中 op_a 是二元算术算子, op_b 是二元逻辑算子, op_c 是二元逻辑比较算子)

图 3 是对流图语言语义的定义, 转移关系 $S \in \mathbb{C} \rightarrow (\Sigma \rightarrow \varphi(\Sigma))$ 可以定义为

$$S(\langle \rho, C \rangle) = \{ \langle \rho', C' \rangle \mid \rho' \in S[act[C]]\rho \wedge suc[C] = lab[C'] \}.$$

定义程序 $P \in \mathbb{P}$ 的转移语义 $S \in \mathbb{P} \rightarrow (\Sigma \rightarrow \varphi(\Sigma))$ 为: $S[P]_s = \{ \langle \rho', C' \rangle \in S(\langle \rho, C \rangle) \mid s = \langle \rho, C \rangle \wedge \rho, \rho' \in \mathfrak{S} \wedge C' \in P \}$.

$\iota \subseteq \Sigma \times \Sigma$, 表示状态 $s \in \Sigma$ 与其可能后继 $s' \in \Sigma$ 的二元传递关系, $\langle s, s' \rangle \in \iota$ 当且仅当 $s' \in S(s)$, 简记为 $s \downarrow s'$, ι 是转移关系 S 的另一种表达. 如果命令 $\Lambda, K \in \mathbb{C}$, 且有 $suc[\Lambda] = lab[K]$, K 是 Λ 的后继, 为统一标记, 也记为 $\Lambda \downarrow K$.

程序动作: $S[A]_{\rho}: \mathbb{A} \rightarrow (\mathfrak{S} \rightarrow \mathfrak{S})$
$S[skip]_{\rho} = \rho$
$S[V := E]_{\rho} = \rho[V \mapsto A[E]_{\rho}]$
$S[V := ?]_{\rho} = \rho[V \mapsto n]$, 其中 $n \in \mathbb{Z}$
$S[B]_{\rho} = \rho$, 其中 $B[B]_{\rho} = true$
布尔表达式: $B[E]_{\rho}: \mathbb{B} \rightarrow (\mathfrak{S} \rightarrow \mathfrak{S})$
$B[E_1 op_b E_2]_{\rho} = A[E_1]_{\rho} op_b A[E_2]_{\rho}$
$B[B_1 op_c B_2]_{\rho} = B[B_1]_{\rho} op_c B[B_2]_{\rho}$
算术表达式: $A[E]_{\rho}: \mathbb{E} \rightarrow (\mathfrak{S} \rightarrow \mathfrak{S})$
$A[x]_{\rho} = \rho(x)$
$A[n]_{\rho} = \rho(n)$
$A[E_1 op_a E_2]_{\rho} = A[E_1]_{\rho} op_a A[E_2]_{\rho}$

图 3 流图语言的语义

几个使用到的基本语义函数:

$lab[L_1; A \rightarrow L_2;] = L_1$	$act[L_1; A \rightarrow L_2;] = A$
$suc[L_1; A \rightarrow L_2;] = L_2$	$var[L_1; A \rightarrow L_2;] = var[A]$

迹是指状态转移序列, 迹语义使用有限或是无限的状态序列来模型化计算过程, 反映了程序最精

确的含义. 对于定义的转移系统 $T(\Sigma, \Sigma', \epsilon)$. $\Sigma^0 = \{\epsilon\}$ 表示长度为 0 的迹的集合, ϵ 是空迹; Σ^n 表示长度为 n 的迹的集合, $\Sigma^+ = \{\bigcup_{n>0} \Sigma^n\}$ 表示长度大于 0 的迹的集合; $\Sigma^* = \Sigma^+ \cup \Sigma^0$ 表示有限长度迹的集合; $\Sigma^\omega = s_0 \cdots s_n \cdots$ 表示长度为无穷的迹的集合. 有非空迹 $\sigma \in \Sigma^\omega \cup \Sigma^+$, 则迹 σ 可表示为 $s_1 \downarrow s_2 \downarrow s_3 \cdots \downarrow \cdots$. 其中 $s_i \in \Sigma$; $\Sigma' = \Sigma^* \cup \Sigma^\omega$ 表示迹的集合. 对于程序 $P \in \mathbb{P}$ 和初始状态 Σ_i , 迹语义函数 $\Sigma': \mathbb{P} \rightarrow \Sigma'$, $\Sigma'[P] = \{s_0 = \Sigma_i\} \cup \{s s' \mid s s' \in \Sigma' \wedge s \downarrow s'\}$. 下面将使用迹语义来作为抽象解释框架的标准语义的定义.

为方便下文的叙述, 给出几个符号的定义:

(1) 限制 ($|_v$): 对于 $\mathcal{V} \subseteq \mathbb{V}$ 下的环境 ρ , $\rho|_v$ 是环境 ρ 在变量集合 \mathcal{V} 上的限制. 定义为

对于程序 $P \in \mathbb{P}$, $\rho|_v = \{\rho(y) \mid y \in \text{dom}[\rho] \wedge y \in \mathcal{V}\}$, $\text{dom}[\rho]$ 表示环境 ρ 包含的变量集合.

也用在对程序 P 的限制, $P|_v$ 是程序 P 在变量集合 \mathcal{V} 上的限制. 定义为

对于程序 $P \in \mathbb{P}$, $P|_v = \{y \mid y \in \text{var}[P] \wedge y \in \mathcal{V}\}$.

(2) 分支 (\parallel): 若序列 $A = a_1 \downarrow a_2 \downarrow \cdots \downarrow a_m$ 和序列 $B = b_1 \downarrow b_2 \downarrow \cdots \downarrow b_n$ 满足 $a_1 = b_1 \wedge a_m = b_n \wedge (\forall i \in (1, m), \forall j \in (1, n), a_i \neq b_j)$, 那么称序列 A 和序列 B 是分支的, 记为 $A \parallel B$.

4.2 实例化迷惑算法

基于流图语言的定义, 下面来形式地描述需要证明的迷惑算法, 证明它是满足可观察语义等价的.

首先形式化可观察语义定义, 这里使用的是指影响指定变量集合的迹集合.

定义抽象函数 $\alpha_0: \alpha_0(T; \mathcal{V}) = \{\alpha_0(\sigma; \mathcal{V}) \mid \sigma \in T\}$; $\alpha_0(\sigma; \mathcal{V}) = \{\lambda i. \alpha_0(\sigma_i; \mathcal{V})\}$; $\alpha_0(\sigma_i; \mathcal{V}) = \langle \rho|_v, C \rangle$. 表示迹序列 T 中限制到变量集合 \mathcal{V} 上得到的迹序列.

我们选择的例子是 Wang^[2] 的压平算法. 压平算法属于控制流迷惑, 它包含的将控制流图展平的思想成为许多迷惑系统中的核心, 如 Cloak 公司的 Cloakware 技术; Wang^[2] 的安全性框架; Collberg^[1] 的 Sandmark 项目; Li^[12] 的基于函数指针的迷惑技术. 因此, 证明压平算法的有效性是有意义的.

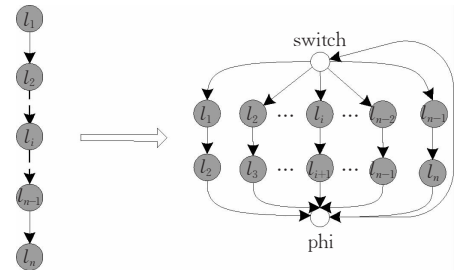
压平算法中, 迟钝变量是其中一个基本部件. 由 Collberg 在文献[1]中提出.

定义 11. 如果变量 V 在程序点 p 具有某种属性 q , q 在迷惑过程中是已知的, 但是在分析时是难以理解的, 这样的表达式称为是迟钝的. 记为 V_p^q , 如果根据上下文程序点 p 是显然的, 则可简写成 V^q . 迟钝变量通常是静态难以推导的值, 但是运行时的值满足一定的性质.

下面给出压平算法的构造思路. 首先定义分支函

数. 设有标签映射对: $\Omega = \{l_1 \mapsto L_1; \cdots; l_n \mapsto L_n\}$. 分支函数 κ_Ω 的作用是实现标签映射对 Ω 中的映射关系, 将到达标签点 l_i 的控制转移跳转到标签 L_i ($1 \leq i \leq n$). 对应到流图语言中, 为实现标签映射对 Ω_1 , 分支函数可以定义为 $\{l_1: A_1 \mapsto L_1; \cdots; l_n: A_n \mapsto L_n\}$.

算法的构造思想是, 使用引入的控制变量来保证变换前后的执行路径一致, 即使得分支函数在变换前后都存在相同的映射. 图 4 给出了压平算法的变换过程, 左边是迷惑前的控制流图, 表示的标签映射对为 $\Omega = \{l_1 \mapsto L_1; \cdots; l_n \mapsto L_n\}$, 右边是迷惑后的控制流图, 同样需要具有与 Ω_1 相同的映射. 通过 $switch$ 节点和 phi 节点的构造来实现. $swVar$ 是其中的控制变量, $swVar$ 的构造通常是迟钝的. 这里由迟钝谓词 V^q 表示, 于是可以得到 $switch$ 节点的构造: $\{switch; swVar = V^{l_1} \mapsto l_1; \cdots; switch; swVar = V^{l_n} \mapsto l_n\}$, $switch$ 节点就等价于分支函数的作用, 构造的分支映射对为 $\Omega_{switch} = \{swVar = V^{l_1} \mapsto l_1; \cdots; swVar = V^{l_n} \mapsto l_n\}$. 因此构造中只需要加入命令来对控制变量 $swVar$ 赋值, 就能保证分支函数具有与 Ω 相同的映射.



(a) 代码迷惑前的控制流图

(b) 压平后的控制流图

图 4 压平算法变换示意图

图 5 给出压平算法的形式描述. 其中 $\kappa_\Omega: \mathbb{L} \mapsto \mathbb{L}$ 是变换前的分支映射函数, 映射对 $\Omega = \{l_1 \mapsto l_2; \cdots; l_{n-1} \mapsto l_n\}$. 为简化流图语言语法, 我们给出的压平算法没有使用复杂数据结构来构造控制变量, 这与 Wang^[2] 的压平算法不同, 但是简单表达式也是一种常用的构造迟钝谓词的方法^[1], 因此这样的变化并不影响对原算法的说明.

Input: 程序 $P(p_1; p_2; \cdots; p_n)$.
 $FL(P) = \begin{cases} (C_{switch} \downarrow p_i \downarrow C_{phi}), & i=1 \\ (C_{switch} \downarrow p_i \downarrow C_{phi} \downarrow C_{skip}) \parallel FL(P - p_i), & \forall i > 1 \end{cases}$
 其中,
 (1) $C_{switch} = (l_{switch} : \{swVar = V^i\} \rightarrow l) \wedge l = lab[p_i]$
 (2) $C_{phi} = (suc(p_i) : \{swVar := V^i\} \rightarrow l) \wedge (l = next(p_i))$
 (3) $C_{skip} = l_{phi} : skip \rightarrow l_{switch}$
 (4) $FL(P - p_i) = P(p_1; \cdots; p_{i-1}; p_{i+1}; p_n)$
 (5) $next(C) = \kappa_\Omega(lab[C])$

图 5 压平算法的描述

4.3 定理的证明

根据我们提出的代码迷惑有效性比较框架, 实例化压平算法的过程还需要证明两个基本的定理: 保证代码迷惑正确性的可观察语义等价性定理以及说明代码迷惑有效性的代码迷惑有效性定理. 限于篇幅, 我们这里只给出定理的形式, 而略去具体证明过程, 这并不影响我们说明建立代码迷惑比较框架的过程, 详细的证明过程可以查阅我们的网站 (http://ssg.ustcsz.edu.cn/publication/jsj06_proof.doc).

定理 1. 压平算法可观察语义等价性. 给定任意 $P(p_1; p_2; \dots; p_n) \in \mathbb{P}$, 满足 $\mathcal{O}[P; \text{var}[P]] = \mathcal{O}[FL(P); \text{var}[P]]$. 即算法 FL 是保 \mathcal{O} 可观察语义等价性.

代码迷惑有效性比较框架的主要目的就是证明迷惑算法的有效性. 因此下面需要证明压平算法的有效性. 基于压平算法的形式化描述, 根据有效性命题 2 可以证明压平算法的有效性定理.

定理 2. 压平算法有效性. 对任意 $\varphi \in uco(\Sigma^*)$, 满足偏序关系 $\varphi(P) \sqsubseteq \varphi(FL(P))$. 即 $FL(P)$ 是针对任意分析算法都是有效的.

这样, 基于我们提出的代码迷惑有效性比较框架, 就完成了对压平算法的有效性证明, 完成了压平算法的实例化过程.

需要注意的是, 为了简化代码迷惑有效性框架的说明, 这里的实例化是针对单个算法, 比较的是单个算法的对语义分析结果的影响, 当然, 也可以使用有效性比较框架来针对不同的算法进行类似地实例化, 从而严格地比较算法之间的有效性.

5 相关工作比较

目前还缺乏从形式语义的角度来进行代码迷惑有效性证明的研究, 一些其它角度的有效性研究包括: Wang 和 Ogiso 采用反迷惑算法的计算复杂度来证明代码迷惑的有效性^[2-3]. 通过证明使用静态分析迷惑后程序的控制流问题可以转化成线性有界图灵机 (LBTM) 可接受问题, 而 LBTM 可接受问题是 PSPACE-complete 的, 从而推导出使用静态分析获得原程序的攻击过程是 PSPACE-complete 的. Barak^[5] 最早从密码分析的复杂性角度来分析代码迷惑的有效性, 建立了代码迷惑的密码分析框架. 该框架的核心部分是: 定义虚拟黑盒属性, 形式化了代码迷惑的要求. Lynn^[6] 推广了 Barak 的结论, 证明了更大一类代码迷惑无法完全保证信息的安全. 这

些证明都不是基于语义模型, 只是分析迷惑算法给反迷惑分析算法带来分析时间上的影响. 类似的, 基于密码理论的迷惑算法有效性研究也只是分析反迷惑算法的分析时间与所花费的控制, 而无法从分析的结果来证明代码迷惑带来的影响.

本文建立了证明代码迷惑有效性的理论框架, 目的是为代码迷惑有效性的比较提供与语言无关的平台, 其中的有效性证明依赖于有效性度量的定义. 关于有效性度量的研究, Dalla^[14-15] 的工作与我们的工作比较接近. 简要比较二者的不同: 首先, Dalla 的工作是针对迹语义建立基于语义的有效性度量, 目的是为了建立有效性度量. 而本文建立了证明代码迷惑的有效性的理论框架, 有效性度量只是有效性比较框架中的一部分, 需要的话, 可以在建立的属性空间上选择不同的有效性度量; 其次, Dalla 使用静态分析模型化反迷惑^[14], 其中一个重要定理是: 如果静态分析得到的元素是格的顶元, 那么称迷惑针对该静态分析是有效的, 根据本文的有效性命题 1 和命题 2 知这个定理是显然的; 最后, 在有效性度量的选择方面, Dalla 扩展了 Collberg 提出的评价迷惑效果的 potency 性质, 提出使用变换前后分析结果的不等来定义有效性度量, 这实际是认为如果变换影响了原本的抽象域就称为有效的, 按照代码迷惑的定义, 理解难度的增加应该是指抽象域发生了偏序变化, 也就是导致分析降低了精度, Dalla 的有效性无法刻画程序变换给语义带来的是降低了精度或是提高了精度, 这样的有效性度量是不准确的. 其它还有许多有效性度量的研究, Collberg^[1] 使用软件工程中评测软件品质的软件复杂性尺度作为度量, 如程序文本中的程序长度, 程序中出现的函数调用数目; 使用程序中语法语义结构的图属性作为度量, 这些图通常是程序语法和语义构造的可视化反映, 如程序控制流图、过程调用图、各种标注信息的依赖图等, 有效性是通过图的属性差异来比较, 如边的总数、图的圈数等.

6 结束语

本文使用抽象解释作为统一的框架来表示迷惑算法和模型化攻击者, 从语义的角度建立了代码迷惑有效性比较框架. 许多研究已经证明了代码迷惑作为安全方法的局限性, 而本文的工作给出了积极的结果, 为证明限定环境下的代码迷惑安全性提供了有益的讨论.

参 考 文 献

- [1] Collberg C, Clark T, Douglas L. A taxonomy of obfuscating transformations. Department of Computer Science, the University of Auckland; Technical Report #148, 1997
- [2] Wang C X. A security architecture for survivability mechanisms[Ph. D. dissertation]. University of Virginia, Department of Computer Science, 2000
- [3] Ogiso T, Sakabe Y. Software obfuscation on a theoretical basis and its implementation. IEEE Transactions on Fundamentals, 2003, E86-A(1): 176-186
- [4] Douglas L. Protecting java code via code obfuscation. ACM Crossroads, 1998, 4(3): 21-23
- [5] Barak B, Goldreich O, Impagliazzo R, Rudich S, Sahai A, Vadhan S P, Yang K. On the (im)possibility of obfuscating programs//Kilian J ed. Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology. Santa Barbara, California; Springer-Verlag, 2001, 19-23; 1-18
- [6] Lynn Benjamin, Manoj P, Amit S. Positive results and techniques for obfuscation//Proceedings of the EuroCRYPT. Interlaken, Switzerland, 2004; 20-39
- [7] Lee B, Larry D. Self-protecting mobile agents obfuscation technique evaluation report. Network Associates Laboratories; Report #01-036, 2002
- [8] Cousot P, Cousot R. Systematic design of program transfor-

- mation frameworks by abstract interpretation//Proceedings of the ACM SIGPLAN Conference on Principles of Programming Languages. Portland Oregon, 2002; 178-190
- [9] Cousot P, Cousot R. Abstract interpretation; A unified lattice model for static analysis of program by construction or approximation of fixpoints//Proceedings of the ACM SIGPLAN on Principles of Programming Languages. Los Angeles, California, 1977; 238-252
- [10] Gregory Wroblewski. General method of program code obfuscation[Ph. D. dissertation]. Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002
- [11] Chen Yi-Yun, Zhang Yu. Theory of Compiler. Beijing: Higher Education Press, 2003(in Chinese)
(陈意云, 张 昱. 编译原理. 北京: 高等教育出版社, 2003)
- [12] Li Yong-Xiang, Chen Yi-Yun. Technique of code obfuscation based on function pointer array. Chinese Journal of Computers, 2004, 27(12): 1706-1711(in Chinese)
(李永祥, 陈意云. 基于函数指针数组的代码迷惑技术. 计算机学报, 2004, 27(12): 1706-1711)
- [13] Cousot P. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. Theoretical Computer Science, 2002, 277(1-2): 47-103
- [14] Dalla Preda M, Giacobazzi R. Semantic-based code obfuscation by abstract interpretation//Proceedings of the ICALP'05. Lisbon, Portugal, 2005; 1325-1336
- [15] Dalla Preda M, Giacobazzi R. Control code obfuscation by abstract interpretation//Proceedings of the SEFM'05. Koblenz, Germany, 2005; 301-310



GAO Ying, born in 1980, Ph. D. candidate. His research interests include theory and implementation of programming language and the security of mobile host.

CHEN Yi-Yun, born in 1946, professor, Ph. D. supervisor. His research interests include theory and implementation of programming language, formal description technologies, and software security.

Background

This research is supported by the National Natural Science Foundation of China(grant No. 60473068). Code Obfuscation is a program transformation for the purpose of increasing the difficulty of program understanding. It is a useful method to guarantee the security in mobile agent and protect the program from reverse engineering. At present, the interests on code obfuscation focus on two major aspects: The construction of code obfuscation algorithm and the proof of its efficiency. Although the construction of code obfuscation algorithm grows more mature, the proof of its efficiency is still blank in the formal semantic foundation. Meanwhile,

many researches have shown the limitation of code obfuscation as a security method, which puts the application of code obfuscation into doubt. Therefore, how to measure and prove the efficiency of code obfuscation is an important problem. The contribution of this paper is to present a new formal framework for proving the efficiency of code obfuscation algorithm. Based on abstract interpretation framework, the authors construct the comparable code obfuscation framework, which can formally prove its efficiency under the limited environment of static analysis and compare the efficiency among code obfuscation algorithms.