

云计算环境约束下的软件水印方案

孙光^{1,2,3}, 樊晓平^{1,3}, 蒋望东³, 周航军³, 刘胜宗³, 龚春红³, 朱静³

- (1. 中南大学计算机科学与技术博士后科研流动站, 湖南 长沙 410083;
2. 美国阿拉巴马大学计算机科学系, 阿拉巴马 塔斯卡卢萨 35401;
3. 湖南财政经济学院财经大数据研究院, 湖南 长沙, 410205)

摘要: 针对云计算环境下的软件知识产权保护需求, 提出一种基于非等价语义混淆理论的软件水印方案。方案设计了一种非等价语义混淆方法, 用于切割、隐藏代码语义, 同时利用混淆规则虚拟映射水印。切割后的语义放置在一个独立的模块中, 验证水印信息、恢复程序正常执行。对该方案的反编译实验表明, 其逆向工程难度大, 保障了水印强健性, 可有效保护放置在云端的代码。

关键词: 云计算; 软件水印; 非等价语义混淆; 强健性

中图分类号: TP391

文献标识码: A

doi: 10.11959/j.issn.2096-109x.2016.00083

Software watermarking scheme with cloud computing constraints

SUN Guang^{1,2,3}, FAN Xiao-ping^{1,3}, JIANG Wang-dong³, ZHOU Hang-jun³,
LIU Sheng-zong³, GONG Chun-hong³, ZHU Jing³

- (1. Computer Science and Technology Postdoctoral Station, Central South University, Changsha 410083, China;
2. Computer Science Department, University of Alabama, Tuscaloosa 35401, USA;
3. Institute of Big Data, Hunan University of Finance & Economics, Changsha 410205, China)

Abstract: A non-equivalent semantic obfuscation-based software watermarking, scheme was introduced to satisfy the demands of software intellectual property breaches in cloud. Via obfuscated source codes, part of semantics was cut into a recovery module to make the source code unavailable. Watermarks map to transform rules were embedded concurrently into these obfuscated codes. The recovery module was a recognizer to prove the watermarks when the original program was recovering. Experimental results, using Java decompilers, indicate that the code become obfuscated, is difficult to reverse engineering, thereby make the watermarks robust.

Key words: cloud computing, software watermarking, non-equivalent semantic obfuscation, robustness

1 引言

Google、微软、亚马逊和 Facebook 等 IT 巨头是云计算的倡导者与先行者, 他们的产品展现

了当前云计算概貌, 是人们了解云计算的重要窗口。这些产品的成败如同云计算的拨号音, 直接影响人们是否接受云计算。但是近年来, 连续不断的云安全事件, 让人们对于云安全普遍感到担忧^[1-4],

收稿日期: 2016-08-18; 修回日期: 2016-09-02。通信作者: 孙光, sunguang@hufe.edu.cn

基金项目: 中美计算机科学研究中心开放基金资助项目 (No.KJR16046); 国家留学基金管理委员会国家公派出国留学基金资助项目 (No.201408430218); 湖南省创新平台开放基金资助项目 (No.16k013)

Foundation Items: Computer Science Research Center of China and The United States Open Fund (No.KJR16046), China Scholarship Council of the Ministry of Education (No.201408430218), Open Innovation Platform Fund Projects in Hunan Province (No.16k013)

严重阻碍了云计算的推广。Google 的 Talk、Gmail, 微软的 Azure、Hotmail、办公在线套件、Amazon Web services (AWS)、Facebook 等, 近年发生多起宕机事件, 造成的损失难以估算。国际咨询公司 BSA (business software alliance) 与 HyperStratus 的调查表明, 超过 45% 的人相信, 现阶段云计算带来的风险大于收益; 超过 80% 的企业提出, 对云计算安全与隐私保护的担忧是阻碍将数字资源放置云端的最大障碍; 超过 90% 的个人认为, 云安全的缺陷让他们近期没有将数字财产放在云端的打算^[5,6]。美国公开发表声明, 准备用 10 年时间缓慢接受云计算, 期望在这 10 年间, 云计算的安全与隐私问题能得到较好的解决。

对于云计算中的隐私保护问题, 很多人相信, SaaS (software-as-a-service) 模式能彻底解决云中软件的盗版问题。因为在云计算模式下, 软件存储在云中, 以服务定制的形式运行, 用户在客户端得到的仅是计算结果, 既接触不到源代码, 也观测不到计算过程, 盗版者与代码被云完全隔开, 这从根本上破坏了软件盗版的基础^[7]。Adobe 公司的 CEO Shantanu Narayen 还补充说, 在 SaaS 模式下运行的软件需要不间断的 Internet 连接, 云平台对运行的软件能实现完全的控制^[8,9], 这使软件盗版非常困难。可根据 BSA 发布的报告, 实际情况远没有这么乐观。近几年, 全球盗版率都维持在 78% 左右, 每年因盗版造成的损失都在增加, 2014 年超过 5 000 亿美元, 其中中国的软件盗版直接损失超过 1 100 亿人民币。更值得注意的是, 将近 10% 的软件已经移植到各类云平台, 这些软件的盗版率及相应的损失基本与传统环境持平, 这说明云计算带给软件防盗版的贡献微乎其微^[4], 云平台上的软件盗版依然猖獗。

云计算的主要业务来自 SaaS, 它的服务质量直接决定了大多数用户对云计算的态度。软件盗版不但威胁了软件产业, 也严重影响了云计算的发展。没有有效的保护措施, 云用户不会冒险将软件系统放置在云平台。因此, 针对云环境的防盗版技术有迫切的需求。软件数字水印属于隐蔽软件技术^[8], 它利用嵌入的信息提供软件版权证明、威慑软件盗版^[9]。近年来, 软件数字水印研

究取得了大量研究成果, 被证明是一种有效的软件保护技术。但该技术的应用背景一直针对传统的计算环境, 直接在云环境中使用是不恰当的^[10]。云环境中的软件盗版行为表现出很多新特点 (将在第 2 节介绍), 软件水印技术应以这些特点为产品背景, 研发更具针对性的软件水印技术, 才能有效保护云中软件的知识产权。本文在分析云环境下软件盗版的基础上, 以云环境作为显约束, 软件数字水印顽健性作为隐约束, 提出一种基于非等价语义混淆的软件水印方案, 探寻云内部软件知识产权保护问题的解。该方案致力于实现软件水印即服务, 在满足显约束的同时, 抵御逆向工程攻击, 保障水印信息的顽健性。

2 云计算环境下的软件应用方式与盗版分析

云中的软件有 2 种应用方式, IaaS (infrastructure-as-a-service) 与 SaaS。如图 1 所示, 在 IaaS 模式下, 提供者按照软件规范, 在本地机或云上 (使用 Platform-as-a-Service 模式) 编制软件、上传到云存储, 用户从云中下载这些软件在本地机上运行^[10]。IaaS 模型中的云相当于远程的公共存储介质, 存放所有用户放置在云端的数字财产^[11]。这种方式与传统计算中的互联网应用模式很像, 也极易遭受传统方式的攻击。云存储 iCloud 作为此次盗版的源头, 与 Internet 上的众多网络用户无异, 黑云作为传播与交易平台, 相当于传统意义上的非法网站或 P2P 网络^[12]。虽然 iCloud 是标准的云计算资源, 但该事件反映出的盗版模式, 与传统计算环境中的盗版没有本质上的区别。

SaaS 模式隔离了最终用户与程序代码, 让两者无法直接接触, 但无论从云平台的外部还是内部看, 这种模式都没能有效解决盗版问题。灰云 (grey cloud) 是指在内部共享登录证书与合法软件的私有云或混合云, 在灰云中存在大量的针对 SaaS 的侵权行为^[13,14]。BSA 的调查报告显示, 超过 42% 的人与他人共享过登录证书, 对付费云服务来说, 这是严重的偷窃。同时登录也是侵权的一种, 很多人都试过同一个 QQ 账号同时在多个设备端 (如电脑、手机、iPad) 登录。由于担心影响云服务的推广, 云提供商不愿意也无法拒绝同时登录, 这就导致此类行为广泛存在。更让人

担忧的是, 超过 21%的组织建立私有云^[4], 这些私有云是黑云与灰云主要的生存空间^[15]。从云的内部来看, SaaS 模式仍然没有解决一个传统的安全问题, who guards the guards?用户对云内安全设置一无所知, 超过 91%的云提供商甚至没有考虑过内部监督^[7], 云内部的盗版监控一直是个空白^[16]。

云平台采用了大量的新技术与新模式, 以保障高性能的公共计算。这些技术在提升云计算效能的同时, 也让软件盗版问题变得更加复杂。1) 为获得高可用性, 云服务往往来自多个云提供商。软件在云中部署时, 需要在不同的云中保存多份副本, 这大大增加了监控软件盗版的难度。2) SaaS 中的服务大都建立在其他服务的基础上, 服务之间构成一种复杂的递归关系。例如, rentalmaps 的 mashup 服务建立在 Google maps services 的基础上。云平台上的海量服务间生成了复杂的引用关系, 让追踪盗版变得异常困难。3) 云中存放着海量软件, 许多软件运行时需要相互合作, 当应用系统间的信息有交叉的时候, 为提高运行效率, 云平台上的虚拟机绕过所有的安全措施, 让实例直接互访^[16], 甚至将消息直接发至另一台虚拟机, 如果攻击者位于云内部, 这种做法对软件的敏感信息是极其危险的^[17], 在软件即服务数据第三方担保 (SaaS data escrow) 没有实施的情况下, 软件的知识产权归属都无法证明。4) 软件栈 (software stack) 是盗版信息的重要来源, 追踪

盗版需要软件栈间的高互操作性。但是云平台至今没有标准化 API, 也没有标准化的计划。这使用户无法提取敏感数据, 更无法利用这些数据判定不同平台上的程序相似性, 极大降低了对盗版软件的发现能力。5) 云软件大都采用许可证方式 (software license) 保护知识产权, 但商业软件在这种模式下无法实施真正意义上的公共计算。许可证方式限制软件的使用与分发, 甚至限制使用的电脑, 而 SaaS 需要利用 Pay-as-you-go 方式提高服务的经济性, 这 2 种方式是不匹配的, 许可证方式很难在云环境下有效保护软件。

软件水印是一种主动保护技术, 它通过在代码中有规律的变换, 映射出代表版权的水印信息^[18,19]。这种主动的信息嵌入能降低发现盗版的成本, 适合复杂环境中的信息识别。在需要版权证明时, 水印提取程序扫描载体中的特征, 判定水印信息是否存在^[20,21]。如图 1 所示, 在云中使用软件水印有 2 种时机。如果用户在本地机上开发软件, 再以 IaaS 方式发布, 那么在软件完成后即可嵌入水印, 把嵌入水印的软件上传到云中。如果软件以 PaaS 模式开发, 那么在发布软件、形成服务之前, 对软件进行嵌入水印处理^[22,23]。云平台上的软件嵌入/提取水印也是一种服务^[24], 可看作 Software Watermarking-as-a-Service。应用该服务时, 既要考虑服务应遵从的云环境限制 (显约束), 也要明确服务对象对它的技术要求 (隐约束)^[25,26]。显约束主要包括: 1) 隐藏保护对象的代码语义,

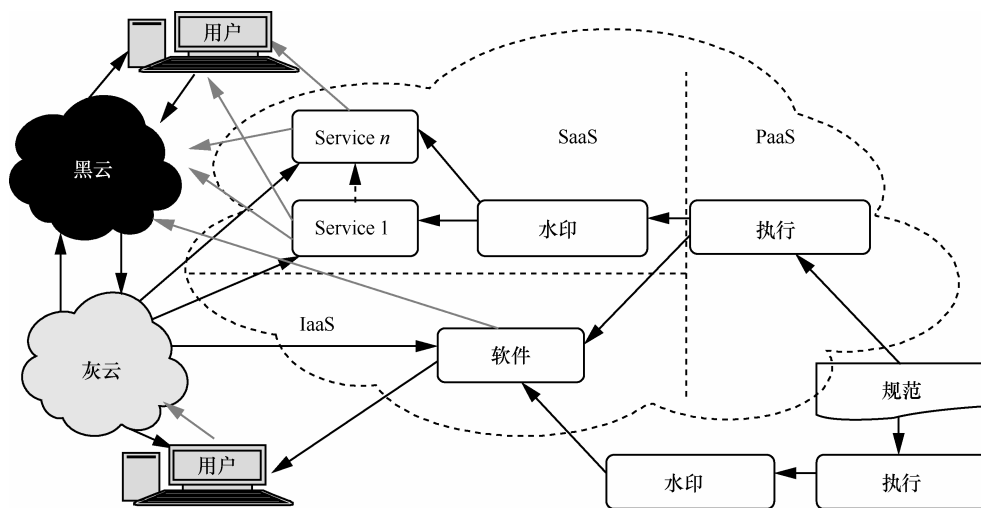


图 1 云计算环境下的软件应用与盗版示意

防止云内部对代码的窃取；2) 软件语义分割，拥有者能控制软件分发与执行。水印方案的隐约束主要是顽健性要求，水印嵌入率与隐秘性并不是该技术的主要衡量指标，否则无法区分水印与信息隐藏技术^[27]。

3 非等价语义混淆与水印嵌入

云计算环境约束下的软件水印方案框架如图2所示，水印信息 WM 为 R 进制数，长度不超过设定的状态数。方案首先根据水印信息生成混淆规则，根据该规则对原程序 p_0 进行混淆，混淆后的代码与原代码在功能上不等价，不能直接运行混淆后的程序，需要在运行前恢复语义。恢复语义的过程也是水印验证的过程，恢复全部语义则验证了水印信息的存在。

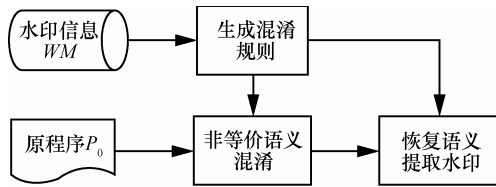


图2 软件水印方案框架

方案中的非等价语义混淆定义如下。

给定一个程序 p ， p 中包含利用源代码定义的对象（如类、方法和语句等）和一个定义在 p 上的代码变换 T （ T 是一个定义在 p 上的非等价语义混淆），如果 $p' \neq T(P)$ ，那么 p' 满足：

- 1) p' 和 p 在输入和执行环境相同时，输出不相同；
- 2) p' 的可理解性远低于 p ，即逆向工程 p' 的时间远大于逆向工程 p 的时间；
- 3) 不存在由 p' 向 p 的逆变换，或者很难构造一个工具自动完成这种逆变换，或者逆变换的代价非常高。

对 p 进行非等价语义混淆是一个顺序的过程，没有并发、竞争、同步等动作，其行为模型是一个典型的有限状态机 FSM (finite state machine)，本文用六元组 $(Q, \Sigma, \Psi, \Delta, \Lambda, q_0)$ 表示。 $(Q = \{q_0, q_1, \dots, q_{m-1}\})$ 为 FSM 状态集合， $m \geq WM$ 的长度， q_0 为起始状态。

$\Sigma = \{C_0, C_1, \dots, C_{n-1}\}$ 为输入集合， C_i 为 p_0 指令集中第 i 条指令， n 为 p_0 指令集指令条数。

$\Psi = \{C_{t_0}, C_{t_1}, \dots, C_{t_{n-1}}\}$ 为输出集合，本方案令 $\Sigma = \Psi$ ，即混淆前后的代码为同一种编程语言。

$\delta_i: \Sigma \rightarrow Q$ 为状态 q_i 下的状态转换函数，规定在状态 q_i 时，某一输入下状态该如何转换。

$\Delta = \{\delta_0, \delta_1, \dots, \delta_{l-1}\}$ 为状态转换函数的集合， $l = mn$ 。

$\lambda_i: \Sigma \rightarrow \Psi$ 为状态 i 下的代码转换函数，规定在状态 i 时，代码如何转换。

$\Lambda = \{\lambda_0, \lambda_1, \dots, \lambda_{l-1}\}$ 为代码转换函数的集合。

图3给出了一个3个状态、4条指令的非等价语义混淆转换，其中

$Q = \{q_0, q_1, q_2\}$ ；

$\Sigma = \Psi = \{add, sub, mul, div\}$ ；

$\delta_0(add) = \delta_0(sub) = \delta_0(mul) = \delta_0(div) = q_1$ ；

$\delta_1(add) = \delta_1(sub) = \delta_1(mul) = \delta_1(div) = q_2$ ；

$\delta_2(add) = \delta_2(sub) = \delta_2(mul) = \delta_2(div) = q_0$ ；

$\Delta = \{\delta_0, \delta_1, \delta_2\}$ 。

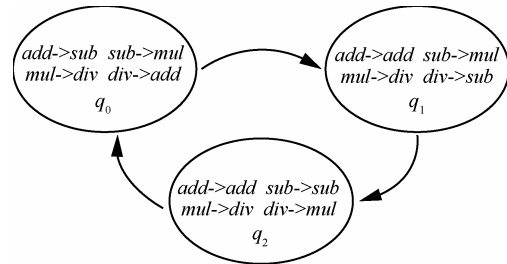


图3 非等价语义混淆与水印嵌入示意

转换函数 λ_i 根据 WM 确定，设 $WM = w_0 w_1 \dots w_k$ ， $0 \leq w_i < R$ 为水印信息第 i 位，规定存在 w_{i-1} 个 λ_{i-1} 函数使 $\lambda_{i-1}(c) = c$ 。如图3所示，为嵌入水印信息“012”，设定

$\lambda_0(add) = sub$ ； $\lambda_0(sub) = mul$ ； $\lambda_0(mul) = div$ ； $\lambda_0(div) = add$ ；

$\lambda_1(add) = add$ ； $\lambda_1(sub) = mul$ ； $\lambda_1(mul) = div$ ； $\lambda_1(div) = sub$ ；

$\lambda_2(add) = add$ ； $\lambda_2(sub) = sub$ ； $\lambda_2(mul) = div$ ； $\lambda_2(div) = mul$ 。

函数 λ_0 将所有指令转换为不同指令，函数 λ_1 中有一条指令不转换，函数 λ_2 有2条指令不转换。

$\Lambda = \{\lambda_0, \lambda_1, \lambda_2\}$

设输入序列为

$add, add, sub, div, mul, add, mul$

FSM 则输出

sub, add, sub, add, div, add, div

为实现该 FSM 功能, 方案需有以下限制。

限制 1 函数 $\delta_i: \Sigma \rightarrow Q$ 是一一对应的, 令反函数为 $\delta_i^{-1}: Q \rightarrow \Sigma$; $\delta_i^{-1}: Q \rightarrow \Sigma$;

限制 2 函数 $\lambda_i: \Sigma \rightarrow \Psi$ 是一一对应的, 令反函数为 $\lambda_i^{-1}: \Psi \rightarrow \Sigma$;

限制 3 将 Σ 分为若干子集, 操作数个数相同的指令放在同一子集中, 限定函数 $\lambda_i: \Sigma \rightarrow \Psi$ 自变量与因变量来自同一子集。

限制 1 和限制 2 确保了语义恢复时, 不存在二义性解释。限制 3 用于在 JVM 或 CLR 上实现时, 避免混淆后的代码操作数的个数与原码操作数的个数不同造成栈平衡错误。

4 非等价语义恢复与水印提取

恢复代码语义不能只建立一个包含 $\delta_i^{-1}: Q \rightarrow \Sigma$ 与 $\lambda_i^{-1}: \Psi \rightarrow \Sigma$ 的反编译器, 如计算

$p: 1+2+3+\dots+n$, 其代码如下。

```
let x=n
let p=0
Loop: if x==0
then exit
else
add p, x
sub x, 1
goto loop
endif
```

实现混淆的 FSM 如下。

$\lambda_0(add) = mul, \lambda_1(sub) = div$

$\delta_0(add) = q_1, \delta_1(sub) = q_2, \delta_1(mul) = q_0$

令指令 if()then...else...endif、goto、let 不转换, q_0 为起始状态, 混淆生成代码如下。

```
let x=n
let p=0
Loop: if x==0
then exit
else
mul p, x
div x, 1
goto loop
endif
```

FSM 对应的反编译器如下。

$\lambda_0^{-1}(mul) = add; \lambda_1^{-1}((div) = sub;$

$\delta_0^{-1}((mul) = q_1; \delta_1^{-1}((div) = q_2;$

$\lambda_2^{-1}((mul) = div; \lambda_3^{-1}((div) = sub;$

$\delta_2^{-1}((mul) = q_3; \delta_3^{-1}((div) = q_4。$

以 FSM 的输出为输入, 反编译器输出。

```
let x=n
let p=0
x==n//第一次循环
add p, x
sub x, 1
x==n-1//第二次循环
div p, x
sub x, 1
```

在输出代码中, 首次循环的起始状态为 q_0 , 结束状态为 q_2 , 而第二次循环起始状态为 q_2 。由于起止状态不同, 同样的循环体代码翻译为不同的语义, 结果显然是错误的。反编译器处理循环体时, 必须每次都有相同的起止状态才能得到正确的结果。有以下 2 种实现方法。

令 $\lambda_0 = \lambda_1 = \lambda_2 = \dots = \lambda_{q-1}$; $\delta(\text{循环体第一条指令}) = \delta(\text{循环体最后一条指令})$

方法 1 令输出函数在任何状态下都相同, 等同于取消了状态转换, 导致混淆强度低, 难以抵御逆向工程攻击。

方法 2 令循环体指令的状态转换为一条回路, 从而确保起止状态相同。对大型软件制定满足所有循环体状态转换为回路的规则是及其复杂的, 本文在循环体中加入一些无实际功能的“傀儡”指令, 用于完成状态的转换。在下列代码中, 指令 $mul\ x, 1$ 没有实际意义, 仅用于将循环体结束状态转换为 q_0 。

```
let x=n
let p=0
Loop: if x==0
then exit
else
add p, x
sub x, 1
mul x, 1//傀儡指令
goto loop
endif
```

设实现非等价语义混淆的 FSM 为 W_x , 反编

译者为 W_x^{-1} 。恢复被混淆的代码语义首先需要建立 W_x^{-1} ，然后在循环体中加入傀儡指令。

W_x^{-1} 也是六元组 $(Q', \Sigma', \Psi', \Delta', \Lambda', q_0)$ ，其中

$Q' = \{q_0, q_1, \dots, q_{m'-1}\}$ 为 W_x^{-1} 的状态集合；

$\Sigma' = \Psi'$ ，为 W_x^{-1} 的输入集合；

$\Psi' = \Sigma$ ，为 W_x^{-1} 的输出集合；

$\Delta' = \{\delta'_0, \delta'_1, \dots, \delta'_{l'-1}\}$ 为 W_x^{-1} 状态转换函数集合；

$\Lambda' = \{\lambda'_0, \lambda'_1, \dots, \lambda'_{l'-1}\}$ 为 W_x^{-1} 翻译函数集合。

q_0 为起始状态；

$m' = m, l' = l$ 。

对 $\forall i, j$ ， $\delta'_i(c_j) = \delta_i[\lambda_i(c_j)]$ ，即 W_x^{-1} 在状态 q_i 输入 C_j 后输出的状态等于 W_x 在状态 q_i 输入 $\lambda_i(c_j)$ 后的状态。

对图 3 所示 W_x ，建立的 W_x^{-1} 如下。

$Q' = Q = \{q_0, q_1, q_2\}$ ；

$\Sigma' = \Psi = \{add, sub, mul, div\}$ ；

$\delta'_0(add) = \delta'_0(sub) = \delta'_0(mul) = \delta'_0(div) = q_1$ ；

$\delta'_1(add) = \delta'_1(sub) = \delta'_1(mul) = \delta'_1(div) = q_2$ ；

$\delta'_2(add) = \delta'_2(sub) = \delta'_2(mul) = \delta'_2(div) = q_0$ ；

$\Delta' = \{\delta'_0, \delta'_1, \delta'_2\}$ ；

$\Psi' = \Sigma = \{add, sub, mul, div\}$ ；

$\lambda'_0(sub) = add$ ； $\lambda'_0(mul) = sub$ ； $\lambda'_0(div) = mul$ ；

$\lambda'_0(add) = div$ ；

$\lambda'_1(add) = add$ ； $\lambda'_1(mul) = sub$ ； $\lambda'_1(div) = mul$ ；

$\lambda'_1(sub) = div$ ；

$\lambda'_2(add) = add$ ； $\lambda'_2(sub) = sub$ ； $\lambda'_2(mul) = div$ ；

$\lambda'_2(div) = mul$ 。

q_0 同样为 W_x^{-1} 的起始状态。 W_x^{-1} 对应的模型如图 4 所示。

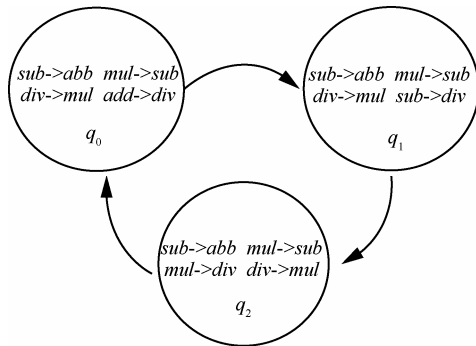


图4 反编译器示意

W_x^{-1} 与 W_x 具有相同的状态转换图，任一状态下的转换则相反。 W_x^{-1} 不能够恢复全部语义，程

序中存在循环指令时，循环体翻译需要相同的起止状态。即对任一状态对 $\langle q_i, q_j \rangle$ ， $\exists$ 指令序列 $\langle d_1, d_2, \dots, d_h \rangle$ ； $h \geq 1$ ，使 W_x^{-1} 在状态 q_i 下输入该序列后，状态由 q_i 转为 q_j 。为此，增加 W_x^{-1} 的限制条件。

限制 4 W_x^{-1} 的状态转换图是连通的，且全局最短路径最短。

分支指令，无论是双分支还是多分支，存在与循环相同的状态转换问题。同一指令中任意 2 个分支的最终状态不同， W_x^{-1} 的翻译也将不同。

为避免此情况发生，本方案设定限制 5。

限制 5 同一指令中的所有分支，必须具有相同的结束状态。

根据本方案，水印提取是通过验证实现的。构造一个能遍历 W_x 或 W_x^{-1} 所有状态的指令序列，依次访问每一状态。状态 q_i 下指令集中未变换的指令条数则为 WM 第 i 位的值。

5 实验结果与分析

非等价语义混淆的实质是对代码的语义分割，混淆形成的代码与源码在功能上不等价，部分语义被切割到了恢复单元。如果攻击者无法获得恢复模块，那么攻击者必需反编译 p_x 才能恢复程序的完整语义，进而避免混淆规则被验证。本文是对 p_x 进行反编译测试，以检验混淆强度和水印顽健性。

对于 p_x 的反编译测试，在硬件方面没有特殊要求，能运行反编译软件即可。实验测试程序为 J2SDK 的范例程序 TicTacToe，随机截取了该程序中的 200 条字节码。实验采用的混淆规则如表 1 所示，构造了 8 个状态，每个状态下转换 8 条指令。若 JAVA 字节码的指令集指令条数为 Jn ，令 $g = Jn - 8$ ，那么嵌入的水印信息 $WM = gggggggg$ 。不同 JAVA 版本中的 Jn 略有差异，大多数版本的指令集都在 200 条左右，本实验采用的 Jasmine 格式指令集有 213 条指令，所以 $g = 205$ 。

实验使用了 20 种反编译工具（见表 2）对混淆后的程序进行反编译测试，反编译结果如表 3 所示。

“混淆前”代表未混淆的程序 p_0 ，“混淆后”指程序 p_x ，符号“×”表示反编译时无法产生 java 文档，“△”表示反编译时可产生 java 文档，但重编译失败，符号“□”表示反编译与重编译均提示成功，

表 1

测试程序输入/输出转换

状态	输入/输出	状态转换	状态	输入/输出	状态转换
q_0	goto / if_icmpne	q_4	q_4	goto / if_icmpne	q_5
	if_icmpne / goto	q_7		if_icmpne / goto	q_4
	iload_1 / iconst_2	q_5		iload_1 / iconst_2	q_2
	iconst_1 / iload_1	q_2		iconst_1 / irem	q_1
	iconst_2 / iconst_1	q_1		iconst_2 / iload_2	q_3
	iadd / irem	q_6		iadd / iload_1	q_0
	iload_2 / iadd	q_3		iload_2 / iconst_1	q_7
	irem / iload_2	q_0		irem / iadd	q_6
q_1	goto / goto	q_1	q_5	goto / goto	q_0
	if_icmpne / if_icmpne	q_3		if_icmpne / if_icmpne	q_1
	iload_1 / iconst_1	q_4		iload_1 / irem	q_4
	iconst_1 / iconst_2	q_2		iconst_1 / iload_2	q_2
	iconst_2 / iload_1	q_0		iconst_2 / iload_1	q_3
	iadd / iload_2	q_6		iadd / iconst_2	q_6
	iload_2 / irem	q_7		iload_2 / iadd	q_5
	iload_2 / irem	q_5		goto / if_icmpne	q_7
q_2	goto / if_icmpne	q_7	q_6	iconst_1 / iconst_2	q_2
	if_icmpne / goto	q_0		goto / if_icmpne	q_5
	iload_1 / iload_2	q_6		if_icmpne / goto	q_4
	iconst_1 / iconst_1	q_5		iload_1 / iload_1	q_1
	iconst_2 / iconst_2	q_4		iconst_1 / iconst_1	q_6
	iadd / iadd	q_3		iconst_2 / iconst_2	q_7
	iload_2 / iload_1	q_1		iadd / iadd	q_0
	irem / irem	q_2		iadd / iload_2	q_3
q_3	goto / goto	q_3	q_7	goto / goto	q_6
	if_icmpne / goto	q_6		if_icmpne / if_icmpne	q_2
	if_icmpne / iadd	q_2		iload_1 / iconst_1	q_5
	iload_1 / iconst_1	q_5		iconst_1 / iadd	q_3
	iconst_1 / iadd	q_1		iconst_2 / iload_2	q_4
	iconst_2 / irem	q_7		iadd / irem	q_1
	iadd / iload_2	q_4		iload_2 / iload_1	q_7
	iload_2 / iconst_2	q_0		iload_1 / iconst_1	q_0

表 2

反编译工具

工具名称	程序开发语言	版本	注释
ocha	Java	Beta 1	the first Java decompiler
SourceTec Decompiler	Java	1.10	Jasmin Java Decompiler a patch of Mocha
Jad	Non-Java, win32	1.5.8e2	
Front End Plus	Non-Java, win32	1.04	Jad engine
DeJava	Non-Java, win32	1.0	Jad engine
Decafe Pro	Non-Java, win32	3.9	Jad engine
CavajJava Decompiler	Non-Java, win32	1.11	Jad engine, trial version
DJ Java Decompiler	Non-Java, win32	3.2.2.67	Jad engine
NMI's Java Class Viewer	Java	5.1	Jadengine, evaluation version
JReversePro	Java	1.4.1	
JODE	Java	1.1.1	
JCavajJava Decompiler	Java	1.0	JODE engine
HomeBrew Decompiler	Non-Java, Unix	0.22	Under development
Dava Decompiler	Java	1.2.5	Soot 1.2.5
Jshrink	Java	2.26	Evaluation version
ClassSpy	Java	2.0	
jAscii	Java	1.0.20	Evaluation version
ClassCracker	Java	3.01	Commercial, demo version
SourceAgain	Unknown	1.1	Commercial, web version
WingDis	Unknown	2.16	Commercial, nodemo version

但重编译后的程序无法正确执行,包括执行程序与原程序在功能上不等价,或者程序资源不匹配。符号“○”表示反编译与执行都正确,恢复的程序在任意输入下与原程序的输出相同。

表3显示,程序混淆前为200条指令,混淆后指令条数为213。混淆后增加的13条指令为傀儡指令,实现循环或分支的指令状态转换,令指令的首尾状态相同。所有反编译工具均可反编译未混淆代码,而对于混淆后的代码,JODE、JCavajJava Decompiler、HomeBrew Decompiler、Class Spy、ClassCracker、SourceAgain、WingDis无法产生合法的.java文档;Jad、Front End Plus、DeJava、Decafe Pro、CavajJava Decompiler、DJ Java、Decompiler、NMI's Java Class Viewer、Jshrink、jAscii无法实现重编译。ocha、SourceTec Decompiler、JReversePro反编译与重编译均显示成功,但对反编译的程序进行重编译后,程序无法运行。只有Dava Decompiler成功反编译了测试代码,且重编译后的程序与原程序在功能上等价。该工具在反编译时采用穷举算法,实验设计的混淆规则有8个状态,每个状态下有8条指令被转换。实验公开状态数与每个状态下转换指令的条数,穷举成功所需次数为 $(8!)^2$,若想加大穷举难度,不公开每个状态下的指令转换条数,则穷举成功需要 $(8!)(200!)$ 次。本方案的水印嵌入率在理论上是无限的,根据水印信息位数设置相应的状态数即可。但方案不可能混淆出无限条数的代码,即使指令条数有限,状态数与混淆规则设计的复杂度也是成正比的。

为对比本方案与已有水印算法在云平台上的适用性,实验测试了现有算法在非等价语义混淆处理后,水印信息的存在情况。由于云平台的虚拟性,云用户必然有代码语义隐藏需求,如果算法ali在非等价语义混淆处理后,还能成功提取水印信息,那么说明ali在隐藏语义后,具有与本方案相同的版权证明能力,ali的水印顽健性,在云环境下不低于本方案。实验测试了Sandmark上所有的10种算法,为论证本软件水印方案的平台无关性,随机选择了sourceforge.net/上的100个java程序作为测试程序,每个程序截取200条字节码,嵌入信息为 $WM = gggggggg, g=205$ 。对任何一种算法,如果能从100个程序中的任何一个成功提取水印,则认为该算法是成功的,实验结果如表4所示。

表3 反编译测试结果

反编译器	混淆前	混淆后	水印
Numberof obfuscated instructions	200	213	—
ocha	O	□	存在
SourceTec Decompiler	O	□	存在
Jad	O	△	存在
Front End Plus	O	△	存在
DeJava	O	△	存在
Decafe Pro	O	△	存在
CavajJava Decompiler	O	△	存在
DJ Java Decompiler	O	△	存在
NMI's Java Class Viewer	O	△	存在
JReversePro	O	□	存在
JODE	O	×	存在
JCavajJava Decompiler	O	×	存在
HomeBrew Decompiler	O	×	存在
Dava Decompiler	O	○	破坏
Jshrink	O	△	存在
Class Spy	O	×	存在
jAscii	O	△	存在
ClassCracker	O	×	存在
SourceAgain	O	×	存在
WingDis	O	×	存在

表4 现有软件水印算法的云环境适用性测试

算法简称	水印技术	水印嵌入结果	水印提取结果
W1	StringConstant	无法嵌入	无法提取
W2	Stern	成功	无法提取
W3	RegisterType	成功	无法提取
W4	Qu/Potkonjak	成功	无法提取
W5	Monden	成功	无法提取
W6	Graph	成功	无法提取
W7	AddMethField	成功	无法提取
W8	AddSwitch	成功	无法提取
W9	Addinitialization	无法嵌入	无法提取
W10	AddExpression	成功	无法提取

表4的测试结果,除了算法W1与W9外,其他算法都成功地嵌入了水印信息。W1与W9为静态软件水印算法,W1通过增加或修改常量定义嵌入水印,W9则利用初始化实现嵌入。这2种算法的水印信息都嵌入在数据段中,不需要修

改代码, 本实验的测试用例只包括可执行代码, 所以这 2 种水印算法无法实施。根据 Collberg 等的实验, 这 2 种算法无法应对任何混淆变换, 是顽健性最弱的算法^[18]。成功嵌入水印信息的算法中, 没有一种算法能在非等价语义混淆后提取水印说明这些算法达不到本方案的顽健性强度。在云平台上的软件, 语义隐藏与代码分割是基本的要求, 上述测试算法还无法达到云计算环境下的软件保护要求。

6 结束语

针对云计算环境下的软件知识产权保护需求, 本文提出一种用云计算环境作为显约束的软件水印方案。该方案利用非等价语义混淆技术隐藏代码语义、嵌入水印信息。由于混淆后的代码与源码在功能上不等价, 语义被切割到恢复模块, 反编译混淆后的代码难度大, 根据本文反编译实验结果, 200 条指令在 8 个状态的转换方案下, 只有穷举法可以破解, 说明方案具有很好的抗逆向工程能力。在实际应用时, 通过增加状态数能有效加大反编译难度。

在云平台上, 恢复模块作为虚拟机的一个插件, 在虚拟机进行二次编译前取得控制权, 实现代码恢复。根据指令集的不同, 本方案有 2 种类型。1) 转换前后的指令集相同, 即把代码混淆为同一种语言; 2) 混淆前后的指令集不同, 通过混淆把原程序转换为不同语言的代码。本方案采用的类型 1, 类型 2 只能在传统的 WIN32 平台上实施, 对于.NET 这样的通用虚拟机, 它支持的所有编程语言只有一种中间码, 类型 2 是无法实施的。从拓展应用面的角度考虑, 本方案应直接在 MSIL 这样的中间码程序上实施, 才能保护该平台上任一编程语言的执行代码。随着 JAVA 与.NET 成为主流开发平台, 对大多数软件用户而言, 实施该方案只需在虚拟机上增加一个纯软件的虚拟机插件, 既不会有太多的技术难度, 也容易控制成本。对于及时性要求高的软件, 须考虑本方案对软件执行效率的影响, 在效率损失和保护强度间有一个科学的平衡。如果本方案在云平台上以服务形式提供, 为不同的用户设计不同的规则, 则能起到良好的保护效果。1) 不同的规则代表着不同的

水印, 水印信息能起到指纹效果, 快速追踪盗版源头; 2) 即使攻击者成功反编译某一恢复模块, 也无法破解其他的模块; 3) 通过控制恢复模块的分发, 防止软件被非法传播。

参考文献:

- [1] Cloud security alliance. Security guidance for critical areas of focus in cloud computing V3.0[EB/OL]. <https://cloudsecurityalliance.org/guidance/csaguide3.0.pdf>.
- [2] Cloud security alliance. Top threats to cloud computing V2.0[EB/OL]. <https://cloudsecurityalliance.org/research/top-threats>.
- [3] Cloud security alliance. The notorious nine: cloud computing top threats in 2015[EB/OL]. <https://cloudsecurityalliance.org/download/the-notorious-nine-cloud-computing-top-threats-in-2015/>.
- [4] Cloud security alliance. Security guidance for critical areas of mobile computing[EB/OL]. https://downloads.cloudsecurityalliance.org%20/initiatives/mobile/Mobile_Guidance_v1.pdf.
- [5] YU Z W, WANG C K, THOMBORSON C. A novel watermarking method for software protection in the cloud[J]. *Software-Practice & Experience*, 2012, 42 (4):409-430.
- [6] NISHIMAKI R, WICHES D, ZHANDRY M. Anonymous traitor tracing: how to embed arbitrary information in a key[C]//Annual International Conference on the Theory and Applications of Cryptographic Techniques. 2016: 388-419.
- [7] BITANSKY N, GOLDWASSER S, JAIN A, et al. Time-lock puzzles from randomized encodings [C]//The 2016 ACM Conference on Innovations in Theoretical Computer Science. 2016: 345-356.
- [8] COLLBERG C, HUNTWORK A, CARTE E. More on graph theoretic software watermarks: implementation, analysis, and attacks[J]. *Information and Software Technology*, 2009, 51(1):56-67.
- [9] HAMILTON J, DANICIC S. An evaluation of the resilience of static java bytecode watermarks against distortive attacks[J]. *International Journal of Computer Science*, 2011, 38(1):90-104.
- [10] KAZUO O, YUANYU W, YUTAKA H, et al. A new watermarking method with obfuscated quasi-chirp transform[J]. *Digital Forensics and Watermarking*, 2012(7128): 57-71.
- [11] ROSADO D G, GÓMEZ R, MELLADO D, et al. Security analysis in the migration to cloud environments[J]. *Future Internet*, 2012, 4(2):469-487.
- [12] GROBAUER B, WALLOSCHKE T, STOCKER E. Understanding cloud computing vulnerabilities[J]. *IEEE Security Privacy*, 2011, 9(2):50-7.
- [13] JANSEN W A. Cloud hooks: security and privacy issues in cloud computing[C]//The 44th Hawaii International Conference on System Sciences. 2011:1-10.
- [14] ZISSIS D, LEKKAS D. Addressing cloud computing security issues[J]. *Futur Gener Comput Syst*, 2012, 28(3):583-592.
- [15] OWENS D. Securing elasticity in the cloud[J]. *Commun ACM*,

2010, 53(6):46-51.

- [16] BISONG A, RAHMAN S. An overview of the security concerns in enterprise cloud computing[J]. International Journal of Network Security & Its Applications, 2011, 3(1): 30-45.
- [17] HWANG K, LI D. Trusted cloud computing with secure resources and data coloring[J]. IEEE Internet Computing, 2010, 14(5): 14-22.
- [18] 张立和, 杨义先, 钮心忻, 等. 软件水印综述[J]. 软件学报, 2008, 14(2): 268-27.
- ZHANG L H, YANG Y X, NIU X X, et al. Review of software watermarking[J]. Journal of Software, 2008, 14(2): 268-27.
- [19] NAGRA M, CHRONI J, NIKOLOPOULOS S D. An efficient graph codec system for software watermarking[C]//Computer Software and Applications Conference (COMPSACW2012). 2012: 595-600.
- [20] COLLBERG C, THOMBORSON C. Watermarking, tamper-proofing, and obfuscation-tools for software protection[J]. IEEE Transactions on Software Engineering, 2012, 28(3): 735-746.
- [21] COLLBERG C, THOMBORSON C, GREGG M. Dynamic graph-based software watermarking[J]. ACM Transactions on Programming Languages and Systems, 2007, 29(6): 77-89.
- [22] ARBOIT G. A method for watermarking java programs via opaque predicates[C]//The 5th International Conference on Electronic Commerce Research (ICECR-5). 2012: 102-106.
- [23] KAMEL I, ALBLUWI Q. A robust software watermarking for copyright protection[J]. Computers & Security, 2009, 28(2): 395-409.
- [24] 芦斌, 罗向阳, 刘粉林. 一种基于混沌的软件水印算法框架及实现[J]. 软件学报, 2007, 18(2): 351-360.
- LU B, LUO X Y, LIU F L. An algorithm framework and realization based on chaos software watermarking[J]. Journal of Software, 2007, 18(2): 351-360.
- [25] DAI P F, WANG C K, YU Z W, et al. A software watermark based architecture for cloud security[J]. Web Technologies and Applications, 2012(7235): 270-281.
- [26] YU Z W, WANG C K, THOMBORSON C. A novel watermarking method for software protection in the cloud[J]. Software—Practice & Experience, 2013, 32(12): 114-129.
- [27] AKRAM M, ADAMU A. Steganographic software: analysis and implementation[J]. International Journal of Computers and Communications, 2012, 6(1): 35-42.

作者简介:



孙光 (1972-), 男, 山东金乡人, 博士, 湖南财政经济学院副教授, 主要研究方向为网络信息安全、大数据图分析与可视化。



樊晓平 (1961-), 男, 浙江绍兴人, 博士, 湖南财经学院副校长、教授、博士生导师, 主要研究方向为大数据科学。



蒋望东 (1971-), 男, 湖南永州人, 硕士, 湖南财政经济学院副教授, 主要研究方向为大数据分析 with 数据安全。



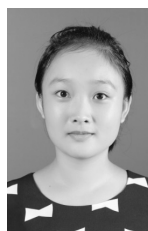
周航军 (1979-), 男, 湖南长沙人, 博士 (后), 湖南财政经济学院副教授, 主要研究方向为云计算与云安全。



刘胜宗 (1986-), 男, 湖南长沙人, 博士, 湖南财政经济学院讲师, 主要研究方向为云计算安全与大数据工程。



龚春红 (1972-), 女, 湖南邵阳人, 硕士, 湖南财政经济学院副教授, 主要研究方向为云计算安全。



朱静 (1995-), 女, 湖南永州人, 湖南财政经济学院本科生, 主要研究方向为云计算安全。