基于改进预处理PCA算法的代码混淆分析

Obfuscation Analysis of IOS Code based on Improved PCA Algorithm

★ 张智华，王勇 上海电力大学
★ 李超 国网数字科技控股有限公司

摘要： 针对各操作系统下不同编程语言的特点导致代码分析结果不够准确的问题，本文结合PCA（principal components analysis）算法在IOS系统下对应用的混淆代码进行分析，其中混淆方式包括控制流混淆、数据混淆和不透明谓词混淆等。文中对已有的PCA代码混淆分析在数据预处理阶段根据实际开发场景进行改进，经实验验证，改进预处理后的PCA算法其KMO值提升了27.7%，证明改进后的数据更适合进行主成分分析且更具有代表性。

关键词： 主成分分析；代码混淆；IOS

Abstract: To address the issue of inaccurate code analysis results are not accurate enough due to the differences between programming languages in various operating systems, this paper analyzes the obfuscated codes of applications in the IOS system by combining the PCA algorithm. The obfuscation methods include control flow confusion, data confusion, and opaque predicate confusion, among others. In this paper, the existing PCA code confusion analysis is improved in the data preprocessing stage according to the actual development scenario. Experimental verification of the KMO value of the improved PCA algorithm after the preprocessing proves that the enhanced data is more appropriate for principal component analysis and more representative.

Key words: PCA; Code obfuscation; IOS

1 引言

后互联网时期信息安全已经成为众多学者关注的要点，对代码进行保护是互联网安全领域中极其重要的部分，对于代码的静态和动态的混淆技术对逆向工程的防御已经相对成熟，对于软件知识产权的保护和各类非法的反编译手段都起到了一定的防护作用。然而对于代码混淆后抵御攻击能力的有效量化分析方法还尚为欠缺，因此代码的有效性评估亟待解决。

目前评估代码混淆有效性的研究主要从代码静态属性复杂度度量和动态运行时攻击度量两个方面出发。基于代码静态属性复杂度度量的研究，包括高鹰等人提出基于语义的代码有效性度量方法，并给出了基于抽象解释的代码混淆有效性的实现框架[1,2]，传统的代码混淆文献[5]对布局混淆、控制流混淆、数据混淆和预防性混淆进行了解释和说明。对于传统的代码混淆分析赵玉洁等人对不透明谓词混淆算法进行了效率和复杂度的分析得出了相关的计算方法[3]。对于代码复杂度的评估，林水明等老师则是对程序容量、圈复杂度、占用内存等参数结合深度学习算法主成分分析的方式得到了相关结果，证明降维后的数据可以有效代表代码的相关属性[4]。文献[6,7]对代码的不同属性进行了专一性讨论，文献[8,9]则是对复合的属性的不同参数对评估结果的影响进行了分析，对于以上提到的评估方式均无法提供综合的评价体系，如果对属性进行简单的累加，必定造成信息的重叠。对于不同的操作系统和不同的编程语言其编译和运行特性也有差异。

针对以上问题本文提出经过预处理改进的主成分分析算法，在IOS系统上对不同形式的代码混淆进行主成分分析并与上文中所提到的主成分方法进行比较验证其优越性。本文的贡献主要如下：（1）对传统的代码混淆PCA算法在预处理阶段进行改进，使参与分析的数据更贴近实际意义并易于分析。（2）提出加权评估算法，对分析得出的数据进行加权评估验证其代码混淆的有效性。

2 相关工作

2.1 代码混淆介绍

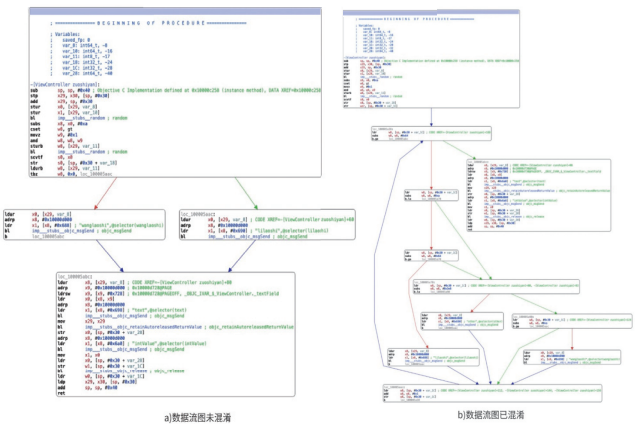
Collberg等在1997年给出代码混淆的明确概念、分类和评价指标。根据混淆所针对的不同属性，将混淆方法分为布局混淆、控制流混淆、数据混淆等，本文通过结合不同方式的代码混淆把其代码作为实验样本。本文实验涉及的代码混淆手段主要如下：

（1）数据混淆

将代码中类名、方法名和属性名替换为无意义符号，增加代码逆向难度。如开源项目ProGuard，能够对Java字节码进行混淆、缩减体积、优化等处理。其中混淆环节就是使用a、b、c、d这样简短无意义的名称，对类、字段和方法进行重命名。此外对于程序中使用的字符串有类如Ollvm提供对字符串进行加密的方法。

（2）控制流混淆

控制流混淆指通过模糊程序代码控制流来混淆程序执行顺序，混淆方法包括不透明谓词、垃圾指令的插入等。文献[10]提出的一种控制流平展方法在类别上就属于控制流混淆，其将串行的控制流通过平展变为并行的控制流，在一定程度上提高了静态分析的难度。文献[11]在LLVM编译器中实现了这种混淆，其通过在中间代码生成阶段添加pass来控制中间代码生成。图1展示了通过控制流平展混淆后的基本块，通过将一个基本块进行拆分、重组，从而达到混淆的目的。



a)数据流图未混淆　　b)数据流图已混淆

图1 数据流混淆前后对比

2.2 主成分分析

主成分分析是多元统计的重要分支之一，以严格的数学理论为基础，通过降维的方法实现多个相关联属性指标的主要信息提取，用少量构造的属性指标解释数据源的大部分信息，构造属性指标即为主成分。本文针对代码混淆分析采用主成分分析原因主要如下：（1）代码混淆分析的要素繁杂，传统的代码分析中涉及的属性不统一，而主成分分析可以通过降维有效地提取贡献率较高的因素来进行分析。（2）基于已有的代码混淆主成分分析对预处理阶段数据进行改进，使分析变量更适合主成分分析且数据更具有代表性。

3 问题描述

目前对代码混淆已有众多学者进行讨论，赵玉洁[3]对于代码的控制流复杂度提出了V（G）= e- n + 2 的设想，在C语言的环境下根据类圈复杂度的算法对代码混淆难度进行评估，另一公式IE=Id/Is则是通过逆向分析工具对平展控制流代码的执行率进行分析。林水明等人则是在JavaScript的代码场景下根据程序容量、圈复杂度、扇入扇出复杂度、参数复杂度、运行时间、占用内存等参数对混淆代码进行主成分分析推导出$COR = \frac{\beta - \alpha}{\alpha}$的算法，证明经过主成分分析降维后的数据和代码混淆难度具有正相关性。以上的研究在实验的验证下对代码混淆在执行效率和效果评估上都作出了贡献，但结合实际的开发场景略有不足。第一，在源代码层面，不同的代码虽然最终都会编译成汇编语言，但在不同的语言中实现相同的算法其代码执行率会有差异，并且在实际的开发过程中使用的大都为面向对象的语言，那么不同的面向对象语言在实现同一功能的编码时编译器也会有所差异。如图2所示，LLVM编译器可以有不同的Frontend，GCC和Clang在编译时的汇编代码也会有所差异。第二，对于已有的主成分分析算法在输入分析项中需要改进，对于上文中提到的主成分分析的样本参数有程序容量、圈复杂度、扇入扇出复杂度、参数复杂度、运行时间、占用内存等。对于一般应用代码，其程序容量巨大，无法进行有效输入。对于同一个模块，在涉及组件化开发的项目中扇入和扇出是不固定的，也无法

准确地给出输入值。对于程序占用内存和运行时间在实际的开发过程中也应具体问题具体对待。对于不同的操作系统同一代码混淆方法入参也存在差异，因此，如需给出更为精确的评估结果需要结合具体情况具体问题具体分析。

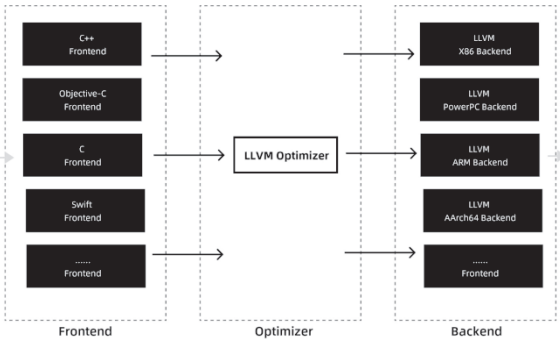


图2 LLVM编译器框架图

针对以上问题本文通过实际代码混淆对IOS系统下的控制流混淆、数据混淆和不透明谓词混淆进行不同入参的主成分分析，在数据预处理阶段根据实际开发场景进行数据提取和处理，使实验数据更具有代表性和针对性，并对实验结果的有效性进行分析。

4 改进的PCA的混淆分析

对于分析过程，我们假设项目中需要混淆的代码为OC（Origin Code），混淆后的代码为OBC（Obfuscation Code）。假设一个模块有N段代码，每段代码的属性有M个，则OC和OBC构成对于上文中提到的主成分分析，目前主要步骤做法是先基于不同的评价指标体系确定要分析的数据，然后进行PCA主成分分析。根据x%方法提取相对应的主成分，然后进行权重的配比，最后通过建立数学模型来算出COR和CAVCC进行代码混淆度评估。以下给出主成分分析的相关定义。

定义1 主成分。设X=（X1，X2，…，Xp）'为p维随机变量，它的主成分矩阵为Y=（Y1，Y2，…，Yp）'，其中第i主成分分量Yi=Ui X（i=1，2，…，p），Ui是正交矩阵U的第i列向量，并满足如下条件：

（1）Y_1是X_1，X_2，…，X_p的线性组合中方差最大者；

（2）Y_k是与Y_1，Y_2，Y_{k-1}不相关的X_1，X_2，…，X_p的线性组合中方差最大者，其中k=2，3，…，p。Y_i也可表示为$Y_i=e'_i X = e_{i1} X_1+e_{i2} X_2 + \cdots + e_{ip} X_p$，其中（$e_{i1}$，$e_{i2}$，…，$e_{ip}$）'是随机变量X的协方差矩阵 Σ的第i个特征值λi所对应的特征向量，且$\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_p \geqslant 0$。

定义 2 主成分决策阵。定义1中的正交矩阵U和（e1，e2，…，ep）' 也称为主成分决策阵。

定义 3 主成分贡献率。假设$\lambda_1\lambda_2$，…，λ_p是随机变量X协方差矩阵Σ的p个特征值，且$\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_p \geqslant 0$，第k主成分的贡献率$C_r$定义为$C_r = \frac{\lambda_r}{\sum_{i=1}^{p}\lambda_i}$，则前k个主成分累计贡献率为$C_{\Sigma r} = \frac{\sum_{k=1}^{r}\lambda_k}{\sum_{i=1}^{p}\lambda_i}$，一般情况下我们取特征值大于1的主成分来进行分析。

4.1 改进思路

本文基于传统的主成分分析代码混淆模型对IOS平台下的商业应用进行数据预处理的改进和分析。改进后的预处理参数如下：

（1）程序容量：已有代码混淆或程序效率研究文章中的程序容量往往都没有明确指明，默认为Demo的代码总容量在大容量代码基数下误差较大，本实验中所涉及到的代码容量计算方式如式（1）：

程序容量 = 混淆方法容量 + 属性容量　　（1）

（2）代码圈复杂度：对开发中同一模块同一代码可能处在不同的上下文中其圈复杂度也不确定的情况，本文结合加权平均数的算法对每种不同的复杂度给予频数作为权值。设同一代码块在组件化程序中有m个不同的上下文环境，那么代码复杂度α的计算公式为式（2）：

$$\alpha = \sum_{i=0}^{m} \alpha_1\theta_1 + \alpha_2\theta_2 + \cdots + \alpha_m\theta_m \quad (2)$$

其中α_i为当前模块i的复杂度，θ_i为当前模块所占的权重。

（3）运行时间：本实验采用打点的方式，具体到开始行和结束行，计算该混淆模块的所有耗时的输入矩阵为$P_{(nxm)}$，在运算中我们根据实际的混淆方法给予m×n维的标准化参数矩阵Z。在标准化过程中根据式（3）：

$$p'_{mn} = \frac{p_{mn} - \bar{p}}{\sqrt{var(p_{mn})}} z_i \quad (3)$$

来对当前的变量进行标准化计算。其整个过程的分析流程，如图3所示。

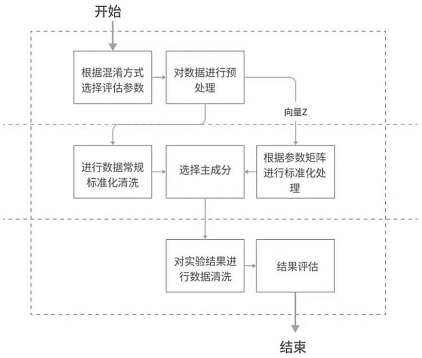


图3 改进的主成分分析结构图

（4）占用内存：因为各操作系统不同版本的运行内存都不相同，本实验的实际内存计算公式为式（4）：

实际内存 = 运行内存 - 静态内存　　（4）

通过实际情况的运行内存差值可以准确估计出当权混淆代码所耗费的内存，也可以减少由于操作系统差异带来的内存误差。

关键步骤如下：

· 实验参数选取：选取实验相关参数主要参考的依据是在程序运行过程中能尽量体现程序性能和反应代码复杂性的指标。根据上文我们优化过的参数最终选取程序容量、代码圈复杂度、运行时间、占用内存、代码行数、代码效率以及代码扇入扇出比等经过预处理的数据作为入参。

· 选取合适的主成分：对于主成分的选取本文采取特征值及贡献率总和的方式来进行选取，原则上主成分的贡献率之和要大于80%，同时主成分的特征值均大于1。

· 实验数据清洗：对于主成分分析得到的数据根据实际情况进行清洗，对于不同的主成分给予不同的权重进行计算。

· 算法性能分析：根据改进算法的实验得出的结果进行分析，主要从主成分贡献度、KMO值和代码复杂度几个指标着重进行讨论。

4.2 算法实现

本文通过提出混淆评估系数（Code Obfuscation Assessment Factor，CAF）和的概念来综合对代码混淆的有效性进行量化和评估。

定义 4 混淆评估系数。对代码$\forall OC_i \in OC$和混淆后的代码$\forall OBC_i \in OBC$，通过基本主成分分析根据x%准则提取前t个主成分，使$C_{\Sigma t} \geqslant x$且$C_{\Sigma(t-1)} < x$，标准化向$z_i \in Z$，Z_i为第i个变量的标准化系数，则有$M=CAF(U,Z)=\sum_{i}^{N}=_1 U_i \oplus Z_i$，其中X为代码块，Z为标准化向量。CAF可以反映出代码混淆后的安全系数。

基于图3改进主成分分析结构图和CAF评估本文进行了算法实现如下：

Start

Assessment Index P_1，P_2，P_3，……，P_n

Obfunctor Input :

OC =（OC_1，OC_2, OC_3, …… , OC_n）

For OBC_i = OBC → Obfunction (OC)

Output Code:

OBC =（OBC_1 , OBC_2 , OBC_3 , …… , OBC_n）

X=（x_1，x_2，x_3...x_n）

Selector Input: X =（X_1，X_2，X_3，....，X_n）

Preprocessor: X_1，X_2，X_3....，X_n

Output: X_p =（x_{p1}，x_{p2}，x_{p3}，....，x_{pn}）

Standardization Input : Z =（Z_1，Z_2，Z_3，....，Z_n）

Standardization : { i = 0 ; Loop i + + ; until z_i == null ;

$p'_{mn}=\frac{p_{mn}-\bar{p}}{\sqrt{var(p_{mn})}}$ z_i（i = 1，2，…，n）;

Output: (P1，P2，…，Pn; (i≤n)) ; }

PC Analysis Selector: { t = 0 ; Loop t + + ; until $C\Sigma t \geqslant x$; Output: (u_1，u_2，…，u_t ; (t≤p)) ; }

end

CAF Assessment Index (U,Z)

Analysis Input:

U =（u_1 , u_2 , u_3 , ... , u_n）

Z =（Z_1 , Z_2 , Z_3 ,.... Z_n）

{ i = 0; Loop t + + ; until i ⩾ n ;

$M_i = CAF(U,Z) = \sum_{i}^{N}=_1 U_i \oplus Z_i$

Output:（M_1，M_2，…，M_n) ; }

end

5 实验结果及其分析

根据第4节中所提到的代码混淆和分析评估

方法，本文基于IOS平台的Object-C语言，以在GitHub上提取的热门程序代码进行混淆作为用例，分别进行了变量名替换、控制流混淆和不透明谓词混淆等手段。基于混淆的结果进行主成分分析并与传统的主程序分析方法进行对比。

5.1 实验数据和环境

本文实验的代码主要来自GitHub上的IOS开发者的Object-C代码，我们挑选了585套不同数量级Star的代码作为实验的原始输入数据并抽取关键部分进行混淆，得到相关的混淆代码作为混淆数据集实验分析的数据源进行分析，其中原始代码的内容展示如表1所示，实验环境如表2所示。

表1 代码混淆数据

代码收藏数	混淆方式	样本集
0-20	变量名替换、控制流混淆、不透明谓词混淆等	30
21-50	变量名替换、控制流混淆、不透明谓词混淆等	49
51-100	变量名替换、控制流混淆、不透明谓词混淆等	215
100 - 500	变量名替换、控制流混淆、不透明谓词混淆等	187
5000 - 10000	变量名替换、控制流混淆、不透明谓词混淆等	89
10000+	变量名替换、控制流混淆、不透明谓词混淆等	15

本文进行的实验在以下环境中进行。

表2 实验环境配置

实验环境	环境配置
操作系统	macOS Big Sur Version（11.4）
处理器	2 GHz 双核Intel Core i5
内存	16 GB 1867 MHz LPDDR3
IDE	XCode（Stable）Version 13.2.1（13C100）
显卡	Intel Iris Graphics 540 1536 MB

根据上述实验数据和环境，我们对IOS应用的代码进行数据分析，并进行矩阵输入得出实验结果。

5.2 实验结果评估

实验输入矩阵包括程序容量、圈复杂度、参数复杂度、运行时间、占用内存、代码行数、代码效率、（入\出）度比等见表3。

表3 经过预处理的输入数据（部分）

代码(4行1组)	程序容量	圈复杂度	参数复杂度	运行时间	占用内存	代码行数	代码效率	入度/出度
1	355	6	1	325	210	548	0.95	0.67
2	378	6	2	353	235	599	0.94	0.67
3	397	6	2.5	366	242	568	0.91	0.67
4	544	16	3.5	432	962	1058	0.65	0.67
5	671	24	2.5	466	678	921	0.73	0.67
6	750	9	4.5	800	1048	985	0.72	0.67
7	781	9	3.5	692	988	1024	0.68	0.67
8	689	24	5.5	900	1132	911	0.71	0.67
9	124	4	1	200	512	208	0.95	0.8
10	162	4	2.5	255	548	244	0.82	0.8
11	187	4	3	288	612	278	0.81	0.8
12	195	6	3.5	353	755	452	0.65	0.8
13	451	9	4.5	466	1103	921	0.54	0.8
14	522	16	4	800	1048	985	0.48	0.8
15	781	13	5.5	752	988	1024	0.68	0.8
16	672	12	4.5	651	956	1052	0.71	0.8
17	224	5	1	203	512	208	0.94	0.8
18	262	5	2.5	242	548	244	0.83	0.8
19	287	5	3	296	612	278	0.79	0.8
20	295	7	3.5	387	755	452	0.75	0.8
…	…	…	…	…	…	…	…	…

对经过处理的数据，我们采用PCA主成分分析得到如下结果。

表4 KMO 和 Bartlett 的检验

KMO值		0.852
Bartlett 球形度检验	近似卡方	338.333
	df	28
	p 值	0.000

从表4可以看出：KMO为0.852，大于0.7，满足主成分分析的前提要求，意味着数据可用于主成分分析研究。数据通过Bartlett球形度检验（$p<0.05$），说明研究数据适合进行主成分分析。我们根据特征值大于1和x%法则提取主成分，提取到两个主成分特征根为5.512和1.374，如表5所示。

表5 PCA方差解释表格

	特征根			主成分提取		
编号	特征根	方差解释率%	累积%	特征根	方差解释率%	累计
1	5.512	68.895	68.895	5.512	68.895	68.895
2	1.374	17.180	86.075	1.374	17.180	86.075
3	0.416	5.194	91.269	-	-	-
4	0.290	3.630	94.900	-	-	-
5	0.153	1.908	96.808	-	-	-
6	0.112	1.406	98.214	-	-	-
7	0.093	1.167	99.381	-	-	-
8	0.050	0.619	100.000	-	-	-

载荷系数可以直观地看出提取后每一个主成分和入参之间的关联性。实验的载荷系数如表6所示。

表6 载荷系数表格

名称	载荷系数		共同度（公因子方差）
	主成分1	主成分2	
程序容量	0.882	-0.231	0.832
圈复杂度	0.827	-0.305	0.777
参数复杂度	0.895	0.219	0.849
运行时间	0.937	-0.113	0.890
占用内存	0.903	0.309	0.911
代码行数	0.946	-0.180	0.928
代码效率	-0.810	-0.382	0.802
出入度比	-0.055	0.945	0.896

通过表5和表6，可以得出如下结论：对于已经提取的特征值大于1的两个主成分累计贡献率为86.075%，一般我们认为PCA的累计贡献率要达到70%以上证明该实验的降维效果符合预期。对于已经提取的主成分，它们对应的加权后方差解释率，即权重，依次为：68.895/86.075 = 80.04%、17.180/86.075 = 19.96%。对于载荷系数，一般来说当载荷值大于0.4时就可以认为当前项和主成分有对应关系。从表6可以看出除代码效率外，程序容量、圈复杂度、参数复杂度、运行时间、占用内存、代码行数和入/出度比都可以较好地由主成分代替。

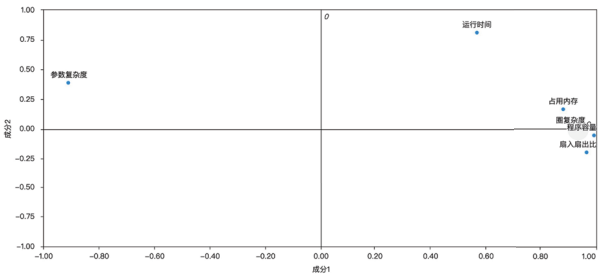
5.3 实验性能分析

本次实验是基于已有的PCA分析针对IOS的平台特性进行的实验，实验在预处理方向根据实际的开发情况做了改进，相对于未做实验处理的数据我们得到了相应的实验数据并进行了对比。上文中所提到的主成分分析其输入参数为程序容量、扇入、扇出、圈复杂度、参数复杂度、运行时间和占用内存，经过基本主成分分析后得到的参数如表7所示。

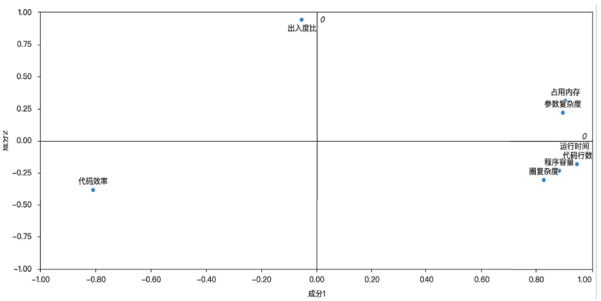
表7 未改进的主成分分析

KMO和Bartlett的检验		
KMO值		0.667
Bartlett球形度检验	近似卡方	73.804
	df	15
	p 值	0.000

通过对比我们观察到表7中的KMO值为0.667，表4中的KMO值为0.852。与表4相比，表7中的K值提高了27.7%，说明我们对实验数据的选择和预处理的方式更适合主成分分析。对于属性的降维效果我们可以通过载荷图的对比进行观察。未改进的实验输入数据有扇入、扇出、内存占用、运行时间、程序容量、圈复杂度、参数复杂度等，改进后的实验输入参数有代码效率、程序容量、圈复杂度、参数复杂度、运行时间、代码行数和出入度比。改进前实验载荷图和改进后实验载荷图如图4（a）、（b）所示。



（a）改进前实验的载荷图



（b）改进后实验的载荷图

图4 改进前、后实验的载荷图

通过观察可以看出图（a）中有6个入参，图

（b）中有8个入参，且扇入扇出包含了图（a）中的两个参数，所以图（b）中的参数实际比图（a）更具有广泛性，而且在上文中已经介绍图（b）中的参数更精准。根据两图对比可以看出在图（a）中载荷系数相对较低的参数复杂度和扇出复杂度在图（b）中都已经有更高的载荷系数被成分1所代表，所以我们认为图（b）具有更好的代表性。

通过以上的分析可以看出，根据实际情况优化之后的主成分入参，在适应主成分算法方面和成分代表方面均优于已提出的算法，因此我们相信如果根据实际情况深一步推导，可以进行更细致的算法优化，并得到更精确的分析数据。

6 结语

对各操作系统存在的差异性，本文在IOS平台下根据Object-C代码对IOS应用代码做了相应代码混淆实验，并具有针对性地对主成分分析的分析参数进行了具有实际意义的改进，通过改进前后的数据对比得出的结果也具有实际意义。根据本实验结果，我们可以发现代码混淆分析理论和实践的差异性，例如同一套混淆代码在LLVM编译器的Frontend和Backend的不同形式组合下产生的参数也会有所差异，这是不可避免的。

未来要在代码混淆或代码分析方向提出可靠的分析方法，因此需要我们经过大量的实验去探索各系统的差异性才能使分析结果更精准可信。AP

★基金项目：上海市自然科学基金项目“电力SCADA系统假数据注入攻击的关系图谱检测方法研究”（项目编号：20ZR1455900）。

作者简介

张智华（1992-），男，河南焦作人，硕士研究生，现就读于上海电力大学计算机科学与技术学院，主要研究方向为移动应用安全、移动网络安全等。

王　勇（1973-），男，河南驻马店人，教授，博士，现就职于上海电力大学计算机科学与技术学院，主要研究方向为网络安全、恶意代码检测、电力信息安全等。

李　超（1986-），男，山东泰安人，学士，现就职于国网数字科技控股有限公司，主要研究方向为电力信息安全、电力安全监测等。

参考文献：

[1] Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]. Proc of IEEE Conference on Computer Vision and Patter Recognition, 2005 : 1 - 10.

[2] 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.

[3] 赵玉洁, 汤战勇, 王妮, 等. 代码混淆算法有效性评估[J]. 软件学报, 2012, 23 (3) : 700 - 711.

[4] 林水明, 吴伟民, 陶桂华, 林志毅, 苏庆. 基于主成分分析的代码混淆有效性综合评估模型[J]. 计算机应用研究, 2016, 33 (09) : 2819-2822+2840.

[5] Collberg C, Thomborson C, Low D. A taxonomy of obfuscating transformations[R]. Department of Computer Science, The University of Auckland, 1997.

[6] Li Haoxiang, Lin Zhe, Brandt Jetal. Efficient boosted exemplar based face detection[C]. Proc of IEEE Conference on Computer Vi- sion and Patter Recognition, 2014 : 1843 - 1850.

[7] Tu Zhuowen. Probabilistic boosting-tree: learning discriminative models for classification, recognition and clustering[C]. Proc of IEEE Conference on International Conference on Computer Vision, 2005 : 1589 - 1596.

[8] Bertholon B, Varrette S, Martinez S. ShadObf: a C-Source obfuscator based on multi-objective optimisation algorithms[C]. Proc of IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, 2013 : 435 - 444.

[9] Bertholon B, Varrette S, Bouvry P. JShadObf: a JavaScript obfuscator based on multi-objective optimization algorithms[C]. Proc of IEEE Internatronal Conference on Network and System Security. [S.l.]: IEEE Computer Society, 2013 : 31 - 41.

[10] 李路鹿, 张峰, 李国繁. 代码混淆技术研究综述[J]. 软件, 2020, 41 (02) : 62 - 65.

[11] Bertholon B, Varrette S, Bouvry P. Comparison of multi-objective optimization algorithms for the JShadObf JavaScript obfuscator[C]/ / Proc of IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. 2014 : 489 - 496.