Journal of Computer Applications
计算机应用,2017,37(6):1803 - 1807
ISSN 1001-9081
CODEN JYIIDU
2017-06-10
http://www.joca.cn

文章编号:1001-9081(2017)06-1803-05
DOI:10.11772/j.issn.1001-9081.2017.06.1803

基于同余方程和改进的压扁控制流的混淆算法

王　岩[1,2,3],黄章进[1,2,3*],顾乃杰[1,2,3]

(1.中国科学技术大学 计算机科学与技术学院,合肥 230027;　2.中国科学技术大学 安徽省计算与通信重点实验室,合肥 230027;
3.中国科学技术大学 先进技术研究院,合肥 230027)

(*通信作者电子邮箱 zhuang@ustc.edu.cn)

摘　要:针对现有控制流混淆算法的混淆结果单一的问题,提出了一种基于同余方程和改进的压扁控制流混淆算法。首先,使用密钥和一组同余方程来生成源代码的基本块中需要使用的不透明谓词;其次,基于 Logistic 混沌映射提出了一种新的 N 态不透明谓词构造算法,并将其应用到现有的压扁控制流算法中,对现有的压扁控制流算法进行改进;最后,将上述两个对源码进行混淆的算法结合,以此来增加源代码中控制流的复杂度,使其更难被破解。与现有的基于混沌不透明谓词的压扁控制流算法相比,所提混淆算法使混淆后代码的防篡改攻击时间平均提高了 22% 以上,总圈复杂度平均提高了 34% 以上。实验结果表明,所提算法能够保证混淆后程序执行结果的正确性并且具有很高的圈复杂度,能够有效地抵抗静态攻击和动态攻击。

关键词:代码混淆;N 态不透明谓词;同余方程;压扁控制流算法

中图分类号:TP311.56　　**文献标志码**:A

Obfuscating algorithm based on congruence equation and improved flat control flow

WANG Yan[1,2,3], HUANG Zhangjin[1,2,3*], GU Naijie[1,2,3]

(1. *School of Computer Science and Technology, University of Science and Technology of China, Hefei Anhui* 230027, *China*;
2. *Anhui Province Key Laboratory of Computing and Communication Software, University of Science and Technology of China, Hefei Anhui* 230027, *China*;
3. *Institute of Advanced Technology, University of Science and Technology of China, Hefei Anhui* 230027, *China*)

Abstract: Aiming at the simple obfuscating result of the existing control flow obfuscating algorithm, an obfuscating algorithm based on congruence equation and improved flat control flow was presented. First of all, a kind of opaque predicate used in the basic block of the source code was created by using secret keys and a group of congruence equation. Then, a new algorithm for creating N-state opaque predicate was presented based on Logistic chaotic mapping. The proposed algorithm was applied to the existing flat control flow algorithm for improving it. Finally, according to the combination of the above two proposed algorithms for obfuscating the source code, the complexity of the flat control flow in the code was increased and make it more difficult to be cracked. Compared with the flat control flow algorithm based on chaotic opaque predicate, the code's tamper-proof attack time of the obfuscated code was increased by above 22% on average and its code's total cyclomatic complexity was improved by above 34% on average by using the proposed obfuscating algorithm. The experimental results show that, the proposed algorithm can guarantee the correctness of execution result of the obfuscated program and has a high cyclomatic complexity, so it can effectively resist static and dynamic attacks.

Key words: code obfuscation; N-State opaque predicate; congruence equation; flat control flow algorithm

0　引言

近年来随着软件技术的飞速发展,软件代码的安全保护越来越引起人们的重视。为了提高软件的可靠性,代码混淆作为一种抵抗软件逆向分析的方法被提出[1]。代码混淆[2]是指对拟发布的应用程序进行保持语义转换,使得变换后的程序与原来的程序在功能上相同或相近,但更难被理解和反编译。Collberg 等[2-5]将代码混淆分为外形混淆、数据混淆、预防性混淆和控制混淆四类。控制混淆相对于其他三种混淆具有更好的安全性,是当下代码混淆领域主要的研究热点。控制混淆主要依赖于不透明谓词[3]。Arboit[6]表明可以将谓词进行参数化来构造更加复杂的谓词,并提出了一种使用二次剩余构造不透明谓词的方法。但是,Myles 等[7]在其实验中证明了使用二次剩余构造出的不透明谓词在安全性方面表现得很差。袁征等[8]提出了一种基于初等数论里面的同余方程来生成不透明谓词的方法。这种不透明谓词存在形式过于简单、安全性差的缺点,在逆向分析过程中较容易被过滤[1]。苏庆等[1]通过改进 Logistic 混沌映射,提出了一种新

收稿日期:2016-11-15;**修回日期**:2016-12-21。

基金项目:安徽省自然科学基金资助项目(1408085MKL06);高等学校学科创新引智计划项目(B07033)。

作者简介:王岩(1991—),男,山东济南人,硕士研究生,CCF 会员,主要研究方向:软件技术、程序优化;　黄章进(1980—),男,湖北天门人,副教授,博士,CCF 会员,主要研究方向:计算机图形学、图形处理器计算;　顾乃杰(1961—),男,江苏南通人,教授,硕士,CCF 会员,主要研究方向:并行算法、并行处理、并行体系机构。

的混沌映射,使用该映射构造出了一种混沌不透明谓词,但是这种混沌不透明谓词只有在结果为真时才具有相对较高的密码安全性。

Wang[9]第一次提出了基于 switch-case 的控制流压扁算法。这个算法将源代码划分成基本块,将基本块打乱后放入 switch-case 结构中;由 case 中的条件变量来控制基本块的执行顺序,将其按照源码中基本块原来的执行顺序来执行;最后将 switch 封装到死循环中,当执行完最后一个基本块时,退出死循环。吴伟民等[10]在此基础上提出了 N 态二维混沌不透明谓词,将二态不透明谓词扩展成 N 态;然后将其应用于控制流压扁算法中的 switch-case 语句中的 case 常量表达式中。这在一定程度上提高了逆向分析的难度。但是,给 case 中控制下一步 switch 走向的变量赋的是常量值,使其暴露在攻击者的视野中,在一定程度上降低了破解的难度。

本文利用密钥和若干同余方程组解的状态来生成不透明谓词,并将其应用于源代码的基本块中,这种构造方法与陈代梅等[11]提出的使用同余方程和中国剩余定理来构造不透明谓词的方法相比,省去了对生成的多项式进行两两互素的计算,减少了计算时间,且产生的不透明谓词为 True 或 False 的几率基本相同。本文基于分段 Logistic 混沌映射[12]提出了一种新的 N 态混沌不透明谓词的构造算法,并将其与吴伟民等[10]改进的压扁控制流算法相结合,隐藏对 case 语句中控制变量的赋值。依此对源代码进行控制流混淆,混淆后的代码不仅具有很高的安全性,并且具有很高的圈复杂度,能够有效地抵抗逆向工程的攻击。

1 基本概念

1.1 不透明谓词

定义 1 不透明谓词[1]。对于一个谓词 P,如果程序中点 p 的输出在嵌入程序之前就已知,则该谓词 P 是不透明的。如果谓词 P 的输出永远为真,则记为 P^{T};如果谓词 P 的输出永远为假,则记为 P^{F};如果谓词的输出有时为真有时为假,则记为 $P^{?}$。

定义 2 陷门不透明谓词[1]。令 K_j 为谓词 P 的密钥,若 K_j 已知,则混淆前很容易确定谓词 P 在程序中点 p 上的输出;否则若 K_j 未知,则混淆前难以确定谓词 P 在程序中点 p 上的输出,则称谓词为陷门不透明谓词。

定义 3 N 态不透明谓词[10]。对于某一确定的实现机制,不透明谓词表达式 $P = E(O)$ 的可能取值为 $1,2,\cdots,N$,其中 O 为谓词定义域,通过表达式映射 E 所对应的 P 构成了 N 态不透明谓词。

1.2 不透明谓词的插入

在程序中插入的不透明谓词主要有永真不透明谓词、永假不透明谓词、可真可假的不透明谓词。在程序中插入不透明谓词的方法如图 1 ~ 3 所示[11],图中:$B_i(i = 1,2,3)$ 表示程序中的基本块,$f(B_i)$ 表示基本块 B_i 的语义,实线表示可能的执行路径,虚线表示永远不会执行的路径,P^{T} 表示不透明谓词的输出为 True,P^{F} 表示不透明谓词的输出为 False,$P^{?}$ 表示不透明谓词的输出为 True 或 False,$B_{2\mathrm{bug}}$ 表示垃圾代码。不透明谓词的插入方式就是在基本块之间添加一个条件判断语句,根据不透明谓词输出的结果判断执行哪个基本块。

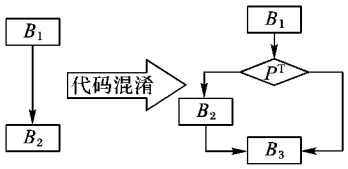

图 1 永真不透明谓词

Fig. 1 Always true opaque predicates

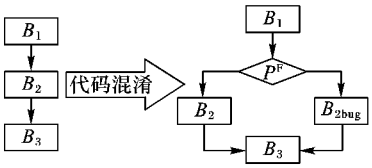

图 2 永假不透明谓词($f(B_2) \neq f(B_{2\mathrm{bug}})$)

Fig. 2 Always false opaque predicates($f(B_2) \neq f(B_{2\mathrm{bug}})$)

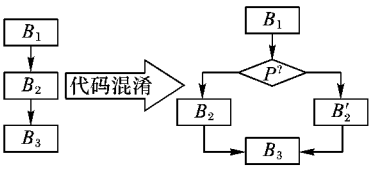

图 3 可真可假不透明谓词($f(B_2) \neq f(B'_2)$)

Fig. 3 True or false opaque predicates($f(B_2) \neq f(B'_2)$)

2 代码混淆算法

本章首先描述不透明谓词的构造算法,然后提出基于分段 Logistic 映射的 N 态不透明谓词的构造算法,最后给出改进后的压扁控制流算法。

2.1 构造不透明谓词

使用 P_j 表示构造出的不透明谓词,本文使用密钥并基于若干组同余方程[13]的解的状态来对不透明谓词 $\{P_j\}_{j=1}^{j=n}(j = 1,2,\cdots,n)$ 进行参数化。

记模素数 p 的 Legendre 符号为 (d/p)。

同余方程[13]如下:

1) 同余方程 1:

$$x^2 \equiv -1 \pmod{p} \tag{1}$$

当 $-1/p = 1$,即 $p = 4k + 1(k \in \mathbf{Z})$ 时有解,设最小整数解为 x_1。

2) 同余方程 2:

$$x^2 \equiv 2 \pmod{p} \tag{2}$$

当 $2/p = 1$,即 $p = 8k + 1$ 或 $p = 8k + 7(k \in \mathbf{Z})$ 时有解,设最小整数解为 x_2。

3) 同余方程 3:

$$x^2 \equiv -2 \pmod{p} \tag{3}$$

当 $-2/p = 1$,即 $p = 8k + 1$ 或 $p = 8k + 3(k \in \mathbf{Z})$ 时有解,设最小整数解为 x_3。

4) 同余方程 4:

$$x^2 \equiv 3 \pmod{p} \tag{4}$$

当 $3/p = 1$,即 $p = 12k + 1$ 或 $p = 12k + 11(k \in \mathbf{Z})$ 时有解,设最小整数解为 x_4。

5) 同余方程 5:

$$x^2 = -3 \pmod p \tag{5}$$

当 $-3/p = 1$,即 $p = 6k+1(k \in \mathbf{Z})$ 时有解,设最小整数解为 x_5。

本文构造不透明谓词的过程如下:

1) 设 $N_j \in \mathbf{Z}^+$ $(j = 1,2,\cdots,n)$,随机生成 N_j 个整数 $(k_1, k_2, \cdots, k_{N_j})$ 为谓词 P_j 的密钥,对于每个整数 k_i,根据式(1) ~ (5) 中判定是否有解的等式(如 $p = 4k+1$),将 k_i 代入其中,至少有一个解 p 是素数。

2) 确定 p 后,根据式(1) ~ (5) 解出 $(x_i)_{i=1}^{i=5}$。当 x_i 有解时记为 1,无解时记为 0,最后将 $(x_i)_{i=1}^{i=5}$ 放入数组中,形成一个五元组 $(x_1, x_2, \cdots, x_5)$。

3) 对于每个 N_j 产生的五元组解,将每一组解中互相对应每一位进行异或操作,最后得到一个五元组解记为 $(r_1, r_2, \cdots, r_5)$。

4) 根据其中 1 的个数来判断不透明谓词的输出,当 1 的个数为奇数时不透明谓词输出为 True,当 1 的个数为偶数时不透明谓词输出为 False。

表 1 ~ 2 给出了本文提出的算法与陈代梅等[11]提出的算法在产生结果为 True 的不透明谓词的个数和产生不透明谓词的时间上的实验结果对比。

表 1　结果为 True 的不透明谓词的个数对比

Tab. 1　Number comparison of creating predicates with true result

算法	产生的不透明谓词个数			
	10	100	1 000	10 000
文献[11]算法	1	31	280	2 827
本文算法	4	47	438	4 510

根据表 1 中的数据,本文提出的算法产生的不透明谓词结果为 True 或 False 的个数基本相同,产生 10 ~ 10 000 个不透明谓词时,结果为 True 的混沌不透明谓词分别占 40%、47%、43.8%、45.1%,生成不透明谓词的个数越多,其百分比越接近 50%,结果为 True 或 False 的混沌不透明谓词的个数越接近。本文提出的算法在生成不透明谓词的均衡性方面优于陈代梅等[11]提出的算法。

表 2　产生不透明谓词的时间对比　ms

Tab. 2　Time comparison of creating predicates　ms

算法	产生的不透明谓词个数			
	10	100	1 000	10 000
文献[11]算法	0.98	9.58	86.11	690.55
本文算法	0.36	2.99	29.31	248.16

由表 2 的数据可知,在产生 10 个 ~ 10 000 个不透明谓词时,使用本文提出的算法所消耗的时间相对于陈代梅等[11]提出的算法所消耗的时间分别降低了 63.38%、68.79%、65.96%、64.06%,其开销小于使用陈代梅等[11]的算法产生的开销。

2.2　基于分段 Logistic 映射的 N 态不透明谓词构造算法

分段 Logistic 混沌映射[12]具有非线性性质,采用此映射生成混沌序列时不需要进行扰动运算,能够保证生成的算法具有更好的效率和安全性。定义如下:

$$a_{n+1} = \begin{cases} 4 \times u \times a_n \times (0.5 - a_n), & 0 \leqslant a_n < 0.5 \\ 1 - 4 \times u \times a_n \times (0.5 - a_n) \times (1 - a_n), & 0.5 \leqslant a_n \leqslant 1 \end{cases} \tag{6}$$

其中,$3.569\,946\cdots \leqslant u \leqslant 4$,$a_0 \in (0,1)$。

以下所述 N 态不透明谓词构造算法就是定义 3 中的实现机制 E,而密钥 (a_0, u, fun) 就是其中的谓词。本文构造 N 态不透明谓词的算法描述如下:

步骤 1　根据式(6) 对参数的要求,使用随机生成的二元组密钥 (a_0, u) 进入混沌系统产生随机实数序列 $A = \{a_1, a_2, \cdots, a_N\}$。

步骤 2　通过映射函数 fun 将实数序列 $A = \{a_1, a_2, \cdots, a_N\}$ 映射成整数序列 $F = \{F_1, F_2, \cdots, F_N\}$,此时添加映射函数后,密钥变成三元组:$(a_0, u, fun)$。

步骤 3　统计 F 中出现的不重复元素的个数 $t(t \in [1, N])$,并将其对应的密钥存放在数组 R 中。

步骤 4　不断重复步骤 1 ~ 步骤 3,训练出与不同 t 值对应的 N 个密钥。存放密钥的数组 $R = \{result^1, result^2, \cdots, result^N\}$,其中 $result^i$ 为步骤 3 中不重复元素个数 t 等于 i 时所对应的密钥 (a_{0i}, u_i, fun),以此类推。

假设 F_i 的取值范围为 $[0, m]$,步骤 2 中使用的映射函数 fun 为:

$$F_i = Round\{a_i \times m\} \tag{7}$$

其中 $Round$ 是取整函数。

2.3　改进的压扁控制流算法

在吴伟民等[10]提出的控制流压扁算法中,对控制变量的赋值为暴露在外的常量值,如算法 1 所示。针对这种情况,本文使用基于分段 Logistic 混沌映射产生的 N 态不透明谓词替换这些常量值,并将 2.1 节中生成不透明谓词的算法应用到程序中的基本块上。改进后的算法如算法 2 所示。

算法 1　文献[10]提出的控制流压扁算法。

```
1)  next = 2;
2)  while (next != 1) {
3)      switch (next) {
4)          case ChaoOpp(ValuesOne):
5)              blockA;
6)              next = 3;
7)              break;
8)          case ChaoOpp(ValuesTwo):
9)              blockB;
10)             next = 1;
11)             break;
12)     }
13) }
```

在算法 1 中,函数 ChaoOpp 是吴伟民等[10]提出的 N 态不透明谓词的生成方法,ValuesOne 和 ValuesTwo 是其生成不透明谓词所需的密钥,blockA、blockB、blockC 是程序中的基本块,在第 1)、6)、10) 行中,对 next 变量赋予的常量值直接暴露在外。

算法 2　本文提出的改进的控制流压扁算法。

```
1)  next = logistic(R1);
2)  while (next != 1) {
3)      switch (next){
```

```
4)      case ChaoOpp(ValuesOne):
5)        if (cPredic(PTrue))
6)          blockA;
7)          next = logistic(R2);
8)          break;
9)      case ChaoOpp(ValuesTwo):
10)       if (cPredic(PFalse)) {
11)         blockBug;}
12)       else {
13)         blockB;
14)         next = logistic(R3);
15)         break;
16)       }
17)   }
18) }
```

在算法 2 中，函数 logistic 是本文基于分段 Logistic 映射产生 N 态不透明谓词的方法，参数 R1 ~ R3 是使用 2.2 节中提出的算法生成的密钥，函数 cPredic 是 2.1 节中提出的生成不透明谓词的算法，其参数 PTrue 和 PFalse 分别是与其对应的生成永真和永假谓词的密钥，blockBug 为插入的垃圾代码的基本块。使用这种算法让程序的控制流程更加难以被分析，安全性更高。

3 实验结果与分析

本文提出的算法均使用 Python 语言实现，并针对几个开源的 Python 程序进行性能测试。

3.1 正确性

对源码进行控制流混淆，首先必须保证其正确性，即混淆后的源码在功能上与混淆前的源码一致，并且拥有相同的输出结果。为了对本文提出的混淆算法进行测试，选了 3 个开源的 Python 工具进行测试，结果如表 3 所示。

表 3 混淆前后程序功能对比

Tab. 3 Program function comparison before and after confusion

被混淆程序	源程序功能	混淆前后功能对比
Pycrypto-master	开源加密工具	功能相同
Docutils	文本转换工具	功能相同
Jieba-master	中文分词工具	功能相同

从表 3 中可以看出，混淆前后的输出结果相同。理论上分析，使用 N 态不透明谓词隐藏程序的控制流，并没有真正改变其基本块的执行顺序，因此并不会影响程序的功能和输出结果。

3.2 安全性

本文使用同余方程生成的不透明谓词以及生成的 N 态不透明谓词，其结果只有在执行的过程中才能确定，即本文生成的不透明谓词是陷门不透明谓词，静态分析并不能确定其输出结果，因此本文提出的生成不透明谓词的算法可以有效地抵抗静态攻击。

由于动态攻击的难点是确定不透明谓词的输出[11]。本文对于基本块中使用的谓词是通过密钥和同余方程的解产生，而 N 态不透明谓词的生成也是根据三元组或四元组密钥产生，且密钥是随机生成的，同余方程解的状态也是随机的，因此可以有效地抵抗动态攻击。

为了对本文提出的混淆算法进行测试，选了 3 个开源的 Python 程序进行防篡改攻击测试，具体统计结果如表 4 所示。

由表 4 中的数据可知，相比使用文献[10]算法，使用本文算法 Pycrypto-master 混淆后产生的攻击时间增加了 28.57%，Docutils 增加了 29.26%，Jieba-master 增加了 22.85%。混淆后代码的攻击时间相比于混淆前大大增加，并且使用本文算法比使用文献[10]算法产生的攻击时间平均高了 22% 以上，使代码变得更加难被篡改。

表 4 代码的防篡改攻击能力对比 h

Tab. 4 Comparison of code's tamper-proof attack ability h

被攻击程序	文献[10]算法	本文算法
Pycrypto-master	2.8	3.6
Docutils	4.1	5.3
Jieba-master	3.5	4.3

3.3 开销

开销主要表现在时间和空间上。时间方面的开销主要是判断不透明谓词的输出，空间方面的开销主要是插入更改控制流的代码。下面分别对表 3 中的开源测试案例进行混淆，并对比其混淆前后在时间和空间方面的开销。

3.3.1 时间开销

通过对表 3 中的开源测试用例进行混淆，混淆前后的时间开销如图 4 所示。从图 4 中可以看出，混淆后程序的执行时间比混淆前要长。其中：Docutils 混淆后较混淆前运行时间增加了 8.08%；Pycrypto-master 增加了 4.04%；Jieba-master 增加了 3.09%。但随着程序的不断增大，增加的时间开销会不断变小，却给程序的破解增加了非常大的难度，因此，这种算法是可取的。

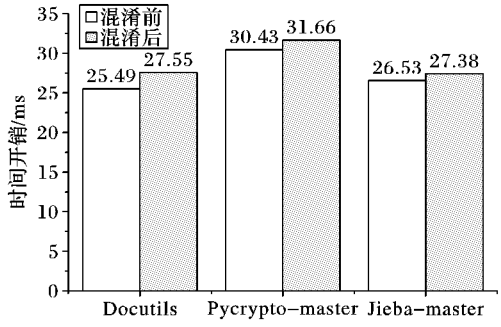

图 4 混淆前后测试用例的时间对比

Fig. 4 Time comparison of test cases before and after confusion

3.3.2 空间开销

在空间开销方面，从图 5 中可以看出，混淆后的程序比混淆前的程序占用的空间增加了。其中：Pycrypto-master 混淆后较混淆前运行空间增加了 10.33%；Docutils 增加了 9.30%；Jieba-master 增加了 0.13%。但随着程序的不断增大，空间开销会越来越小，不会对程序造成很大的负担。

3.4 圈复杂度

现阶段并没有对混淆后程序的复杂度进行评价的统一标准。圈复杂度是衡量一个衡量程序复杂度的度量指标[14]。Radon[15] 可以统计 Python 代码的总圈复杂度。通过多次实验，具体统计结果如表 5 所示。

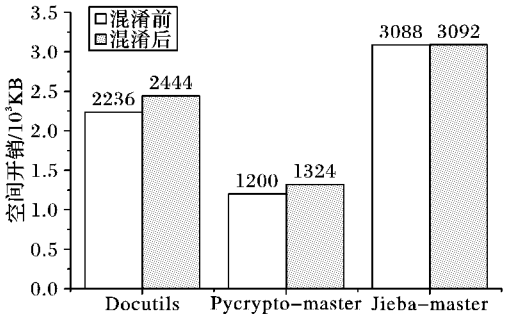

图 5　混淆前后测试用例的空间对比

Fig. 5　Space comparison of test cases before and after confusion

表 5　代码的总圈复杂度对比

Tab. 5　Comparison of the code's total cyclomatic complexity

程序	混淆前	文献[10]算法	混淆后
Pycrypto-master	2083	2817	3935
Docutils	5697	7145	9616
Jieba-master	196	248	337

由表 5 中的数据可知:Pycrypto-master 混淆后较混淆前的总圈复杂度增加了 88.91%;Docutils 增加了 68.79%;Jieba-master 增加了 71.94%。与使用文献[10]算法相比,Pycrypto-master 混淆后产生的总圈复杂度增加了 39.69%;Docutils 增加了 34.58%;Jieba-master 增加了 35.89%。混淆后代码的总圈复杂度相比混淆前大大增加,并且比使用文献[10]算法产生的总圈复杂度平均提高了 34% 以上,代码变得更加复杂。因此,混淆后的代码更难被破解。

4　结语

本文提出了一种简单的基于密钥和同余方程解的状态的不透明谓词生成算法,可大量应用于基本块中。针对当前压扁控制流算法存在的弊端,提出了一种新的 N 态不透明谓词生成算法,并对原有的压扁控制流算法进行改进。最后在正确性、安全性、开销、圈复杂度等方面对本文提出的算法进行了评估。实验结果和分析表明本文提出的算法在正确性和安全性方面表现得很好,具备非常高的圈复杂度,能够有效地抵抗静态攻击和动态攻击。然而,提高代码的混淆度的同时,也增加了时间和空间开销。因此,对于如何在两者之间取得平衡,还需要进一步的研究。

参考文献(References)

[1] 苏庆, 吴伟民, 李忠良, 等. 混沌不透明谓词在代码混淆中的研究与应用[J]. 计算机科学, 2013, 40(6): 155 – 159. (SU Q, WU W M, LI Z L, et al. Research and application of chaos opaque predicate in code obfuscation [J]. Computer Science, 2013, 40(6): 155 – 159.)

[2] COLLBERG C, THOMBORSON C, LOW D. A taxonomy of obfuscating transformations, TR #148[R]. Auckland, New Zealand: University of Auckland, 1997.

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

[4] COLLBERG C, THOMBORSON C, LOW D. Breaking abstractions and un-structuring data structures [C] // ICCL'98: Proceedings of 1998 International Conference on Computer Languages. Piscataway, NJ: IEEE, 1998: 28 – 38.

[5] COLLBERG C S, THOMBORSON C D, LOW D W K. Obfuscation techniques for enhancing software security: US, 6668325 [P]. 2003-12-23.

[6] ARBOIT G. A method for watermarking Java programs via opaque predicates [C/OL] // Proceedings of the 2002 International Conference on Electronic Commerce Research. [2016-10-16]. http://profs.scienze.univr.it/~giaco/download/Watermarking-Obfuscation/sp-paper.pdf.

[7] MYLES G, COLLBERG C. Software watermarking via opaque predicates: implementation, analysis, and attacks [J]. Electronic Commerce Research, 2006, 6(2): 155 – 171.

[8] 袁征, 冯雁, 温巧燕, 等. 构造一种新的混淆 Java 程序的不透明谓词[J]. 北京邮电大学学报, 2007, 30(6): 103 – 106. (YUAN Z, FENG Y, WEN Q Y, et al. Manufacture of a new opaque predicate for Java programs [J]. Journal of Beijing University of Posts and Telecommunications, 2007, 30(6): 103 – 106.)

[9] WANG C X. A security architecture for survivability mechanisms [D]. Charlottesville, VA: University of Virginia, 2001: 65 – 68.

[10] 吴伟民, 林水明, 林志毅. 一种基于混沌不透明谓词的压扁控制流算法[J]. 计算机科学, 2015, 42(5): 178 – 182. (WU W M, LIN S M, LIN Z Y. Chaotic-based opaque predicate control flow flatten algorithm [J]. Computer Science, 2015, 42(5): 178 – 182.)

[11] 陈代梅, 范希辉, 朱静, 等. 基于同余方程和中国剩余定理的混淆算法[J]. 计算机应用研究, 2015, 32(2): 485 – 488. (CHEN D M, FAN X H, ZHU J, et al. Obfuscation algorithms based on congruence equation and Chinese remainder theorem [J]. Application Research of Computers, 2015, 32(2): 485 – 488.)

[12] 王兴元, 朱伟勇. 二维 Logistic 映射中混沌与分形的研究[J]. 中国图象图形学报, 1999, 4(4): 340 – 344. (WANG X Y, ZHU W Y. Researches on chaos and fractal of the coupled Logistic map [J]. Journal of Image and Graphics, 1999, 4(4): 340 – 344.)

[13] 潘承洞, 潘承彪. 简明数论[M]. 北京: 北京大学出版社, 1998: 150 – 162. (PAN C D, PAN C B. Simplified Number Theory [M]. Beijing: Peking University Press, 1998: 150 – 162.)

[14] 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. (ZHAO Y J, TANG Z Y, WANG N, et al. Evaluation of code obfuscating transformation [J]. Journal of Software, 2012, 23(3): 700 – 711.)

[15] LACCHIA M. Radon: a code metrics tool in Python [EB/OL]. [2016-10-16]. https://pypi.python.org/pypi/radon.

This work is partially supported by the Anhui Provincial Natural Science Foundation (1408085MKL06), the Program for Innovation of the Discipline Higher Education (B07033).

WANG Yan, born in 1991, M. S. candidate. His research interests include software technology, program optimization.

HUANG Zhangjin, born in 1980, Ph. D., associate professor. His research interests include computer graphics, graphic processing unit computation.

GU Naijie, born in 1961, M. S., professor. His research interests include parallel algorithm, parallel processing, parallel architecture.