一种基于混沌不透明谓词的压扁控制流算法

吴伟民 林水明 林志毅
(广东工业大学计算机学院 广州 510006)

摘 要 提出了一种基于混沌不透明谓词的压扁控制流算法。该算法将一种新的基于Arnold cat二维混沌映射的N态不透明谓词的构造方法用于改进压扁控制流混淆算法的全局索引变量,并开发了一个基于该算法的JavaScript脚本混淆系统。通过对混淆前后JavaScript程序的静动态分析证明了该混淆算法具有正确性和有效性,同时还能提高混淆后程序的安全性。

关键词 Arnold cat二维混沌映射,N态不透明谓词,压扁控制流算法,JavaScript
中图法分类号 TP309.7 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.5.036

Chaotic-based Opaque Predicate Control Flow Flatten Algorithm

WU Wei-min LIN Shui-ming LIN Zhi-yi
(School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China)

Abstract A control flow flatten algorithm based on chaotic opaque predicate was proposed. This algorithm applies a new construction method of N-state opaque predicates based on Arnold cat planar chaos mapping to improve the global index variable of control flatten flow obfuscation algorithm. A JavaScript obfuscation system was developed depending on this algorithm. Static and dynamic analysis of codes before and after obfuscating proves that the algorithm is correct and effective, and improves obfuscated code's security.

Keywords Arnold cat planar chaos mapping, N-States opaque predicate, Control flow flatten algorithm, JavaScript

1 引言

在软件安全领域,代码保护是其中最重要的策略之一,而基于代码的各类静态和动态混淆技术是实现反逆向工程、保护软件知识产权和取缔各类非法盗版的基本手段。混淆技术大致分为结构混淆、数据混淆、控制混淆和预防性混淆4类[1],而基于控制流分析的混淆技术更受到研究者的青睐。一方面,Wang第一次提出基于switch-case的控制流压扁算法[2],此算法是对源代码的一种等义转换,算法将程序划分成顺序执行的基本块,并把所有基本块放到一个switch语句中,最后将switch语句封装在死循环里。通过更新全局变量的值达到维护正确控制流结构的目的,这样,运行时控制流仍以正确的顺序流经各个基本块,然而程序的控制流结构被彻底地破坏。理论上证明,代码等义转换使得检测代码的问题成为NP完全问题[3]。虽然Collberg提出在程序添加伪别名能够阻止静态分析攻击[4],并给出了基于数组别名的控制流压扁算法[5],但是攻击者仍可通过反汇编进行别名分析;另一方面,袁征等人提出基于数论的不透明谓词混淆方法,在文献中利用二次剩余定理并通过勒让德符号判断产生不透明谓词[6],Arboit利用同余方程构造混淆Java程序的不透明谓词[7],苏庆等提出了基于混沌映射Logistic及其改进的不透明谓词簇的构造方法[8],以上构造不透明谓词的方法已经得到广泛的应用,但其只考虑了二态不透明谓词的构造,即结果只取真或假,而未涉及多态不透明谓词的构造方法。

据此,本文通过二维混沌映射构造N态不透明谓词,并将该方法应用于压扁控制流算法中,提出了基于混沌不透明谓词的压扁控制流算法,并开发了一个基于此算法的JavaScript脚本代码混淆系统。最后,对该算法的正确性、有效性以及代码的安全性进行验证和分析。

2 N态二维混沌不透明谓词

早在1989年,Matthews首次使用混沌映射函数Logistic来设计加密算法[9],2004年Dawei提出一种图像数字水印的算法方案[10],使得混沌理论进入软件安全领域。混沌理论是一种非常复杂的兼具质性思考和量化分析的方法,本文仅借助二维混沌映射产生基于不同初始参数的实数序列。

2.1 二维混沌映射

混沌映射的选择对于不透明谓词构造的影响非常大。张健等提出的改进Arnold cat二维混沌映射函数[11]产生的实数序列具有对初始参数的依赖性、伪随机性以及真值有限性,因此,以该映射为基础所构造的不透明谓词具有很高的安全性能。二维混沌映射表达式为:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = A \begin{bmatrix} x_n \\ y_n \end{bmatrix} \bmod 1 \tag{1}$$

到稿日期:2014-07-15 返修日期:2014-10-01 本文受广东高校优秀青年创新人才培养计划项目(2012LYM_0054),广州市科技计划项目(2012Y2-00046,2013Y2-00043)资助。

吴伟民(1956—),男,教授,硕士生导师,CCF会员,主要研究方向为可视计算、系统工具与平台;林水明(1990—),男,硕士生,主要研究方向为可视计算、代码保护,E-mail:shuiminglin@163.com;林志毅(1979—),男,博士,主要研究方向为代码保护。

当 $A=\begin{bmatrix}1 & 2\\ 1 & 1\end{bmatrix}$ 时，称其为 Armold cat 映射，该映射通过赋予初值 x_0, y_0（$0<x_0<1, 0<y_0<1$），并利用矩阵乘增大实数的值，又通过实数运算模 1 来缩小其值，使得构造出来的实数序列具有混沌特性并取值在[0,1]范围内。因此，三元组 (A, x_0, y_0) 构成了产生混沌序列的密钥。通过选取不同密钥，构造了如图 1 所示的混沌序列空间分布情况，证实了此方法的有效性。

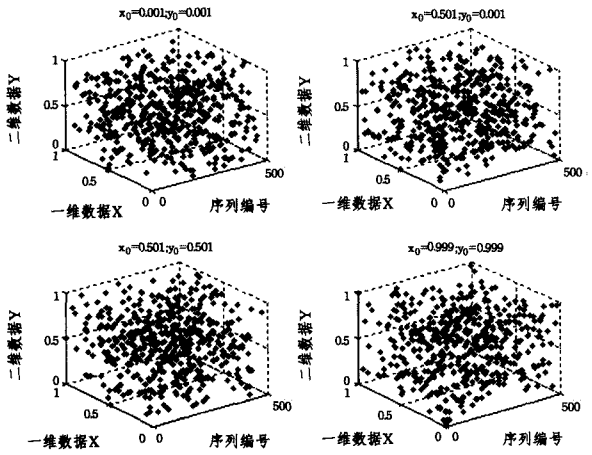

图 1　基于不同密钥的二维混沌序列空间分布图

2.2　混沌不透明谓词构造方法

不透明谓词就是谓词表达式在嵌入程序之前，其真值已经确定。普遍情况下，我们会在程序的分支语句和循环语句中使用布尔类型的谓词表达式作为判断条件，在本文中称之为二态的。在此基础上，本文首次提出 N 态不透明谓词，即构造不透明谓词的结果当且仅当有 N 种取值。下面给出 N 态不透明谓词的定义。

定义 1（N 态不透明谓词）　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。比如以下的算法就是实现机制 E，而密钥 (A, x_0, y_0, fun) 就是其中的谓词。接下来给出构造 N 态不透明谓词的算法描述：

Step1　使用随机函数 *Random* 产生具有 n 个整数的序列构成 $N^*=\{N_1^*, N_2^*, \cdots, N_n^*\}$。

Step2　根据式(1)对参数的要求，使用三元组密钥 (A, x_0, y_0) 进入 Arnold cat 混沌系统产生随机实矩阵序列 $M=\{M_0, M_1, \cdots, M_n\}$，其中 $M_0=[x_0\ y_0]'$，如此类推。

Step3　通过映射函数 *fun* 将实矩阵序列 $M=\{M_0, M_1, \cdots, M_n\}$ 映射成实整数序列，$M=\{M_0, M_1, \cdots, M_n\} \xrightarrow{fun} N=\{N_0, N_1, \cdots, N_n\}$，此时密钥添加映射函数变成四元组：$(A, x_0, y_0, fun)$。

Step4　通过匹配 $(N \xrightarrow{mapping} N^*) \rightarrow result$，$0 \leqslant result \leqslant n$，统计 N 中元素与 N^* 相同的个数存放在 *result* 中。

Step5　不断重复 Step2—Step4，训练出能产生不同 *result* 值的密钥，直至每个取值都有相应的密钥对应。不妨令其为 $\{result^{=1}, result^{=2}, \cdots, result^{=n}\}$，其中 $result^{=1}$ 存放结果恒为 1 的密钥，如此类推。

假设 N^* 的整数范围为 $[0,m]$，本文使用式(2)作为上述所需的映射函数：

$$N_i=Round\{\frac{1}{2}[M(x_i)+M(y_i)]\times m\} \tag{2}$$

其中，$M(x_i)$、$M(y_i)$ 表示二维矩阵 M 每一列的数据，*Round* 是取整函数。

影响训练速度的因素包括整数序列 N^* 元素的范围 m、不透明谓词状态数 n、映射函数以及匹配函数，其中映射函数和匹配函数已经固定。因此，本文对不同 m、n，使用 Matlab 模拟产生不透明谓词所需的迭代次数，结果如图 2 所示。

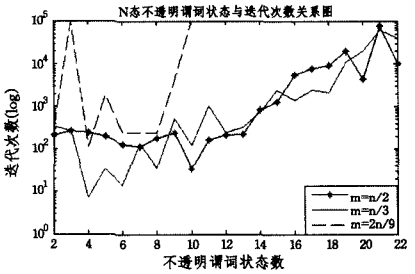

图 2　产生 N 态不透明谓词与迭代次数关系

根据图 2 结果显示，当 $m=n/3$ 时，其迭代效率较高，次数较少，对于产生一个 22 态以下的混沌不透明谓词所需平均迭代次数为 6499，相对 $m=n/2$，$m=2n/9$ 的 6825 和 67907 都要少。并且随着不透明谓词状态数的增加，迭代次数呈指数增长。

3　改进压扁控制流算法

3.1　基于 JS 的算法原型

自从 2002 年 Wang Chenxi 在美国学术会议上提出压扁控制流算法的基本思想以来，各国学者在基于此算法的基础上进行各种变形和改进，并有莫斯科国立大学 Chow 等人给出这种控制流变换有效性的严格证明[12]，直到 2008 年 Akos Kiss 将此算法运用于保护 C/C++代码中，并给出简要算法描述[13]。本文基于前人的研究基础，并结合 JavaScript(JS)脚本语言的特性，将此算法的基本思想应用于保护 JS 代码中。

JS 是一种动态类型、弱类型、基于原型的直译式面向对象脚本语言，其大部分语法与 Java、C++等语言类似，但也有自身特点，具体表现在：JS 通过 eval 方法能够将字符串转换成语句执行；JS 可以使用未声明的变量；JS 代码不需编译直接在带有 JS 支持的浏览器上运行，与具体的平台无关等。其中 eval 方法的字符串参数是执行程序的一部分，将影响程序流程的重构。本文通过将 eval 方法封装成普通语句进行控制流压扁，这样能够忽略 eval 方法的字符串参数作为程序一部分执行对整个程序流程的影响。

基于对 JS 脚本语言特点的分析，我们将源代码分成几类基本块(Block)：循环基本块(包括 while、for 以及 do…while)、分支基本块(包括 if…else 和 switch…case)以及顺序基本块 3 类。据此，基本块在运行中都是顺序执行的。在具体实现的过程中，可以将压扁控制流算法分成 3 类：全局代码控制流压扁、函数体内控制流压扁、基于循环或分支基本块的控制流压扁。其中函数体内的控制流压扁与全局代码的实现方法是一样的，只是多了抽取函数外壳的步骤，这是在代码分析阶段实现的。而分支基本块是循环基本块的特殊形式。以下给出基于全局代码控制流压扁和基于循环基本块控制流压扁的具体算法原型[13]，在这里需要借助 Antlr 进行源码的语

法分析和控制流基本块收集。

```
Initial:nexisiep=1;
loop:while(nextstep!=0){
flatten:switch(nextstep){
case 1:Block「1;nextstep=2;break;
case 2:Block「2;nextstep=3;break;
…
case n-1:Block n-1;nextstep=n;break;
case 「n:Block n;nextsep=0;break;
}}
```

⇩

```
Algorithm「3-1:Global control flow flatten
stringstr:"var i=1;nextstep=1;while (nextstep!=0)";
str:str+"{switch(nextstep){";
loop:string s in block
str:str+"case"+i. tostring()+":"+s;
str:str+"nextstep="+(i+1)%(n+1). tostring();
str:str+"break;";i++;
endloop;
str:str+"}}";
```

图 3　全局代码控制流压扁模型及其算法描述

```
Initial:nexisiep=1;
loop:while(nextstep!=0){
flatten:switch(nextstep){
case 1:if(condition) nextstep=2;
else nextstep=0;break;
case「2:Block1;nextstep=3;break;
case 3:Block 2;nextstep=4;break;
…
case 「n:Block n-1;nextsep=n+1;break;
case 「n+1:Block n;nextstep=1;break;
}}
```

⇩

```
Algorithm 3-2:Loop Block control flow flatten
stringstr:"var i=2;nextstep=1;while (nextstep!=0)";
str:str+"{switch(nextstep){";
str:str+"case 1:if(condition)nextstep=2;"
str:str+"else nextstep=0;break;";
loop:string s in LoopBlock
str:str+"case"+i. tostring()+":"+s;
str:str+"nextstep="+(i+1)%(n+2). tostring();
str:str+"break;";
if(i≥n)i:i+2;else i++;
endloop;
str:str+"}}";
```

图 4　基于循环结构基本块的控制流压扁模型及其算法描述

3.2　改进思路

虽然基于以上算法的代码混淆技术能够有效改变源代码的控制流程，然而对于攻击者来说，通过对 switch 语句中的 *nextstep* 变量进行“使用-定值链”的常量传播分析，可以推算出各个基本块的运行顺序，最后重构控制流图。因此，本文尝试着从 *nextstep* 变量出发，结合以上构造的基于 N 态多维混沌不透明谓词的方法，进一步加强控制流压扁算法，此处的 N 态就是为了自适应变量 *nextstep* 的不同取值，从而实现压扁后代码的正确运行。以下是经过改进的压扁控制流模型。

```
Initial:nexisiep=E^-1(A,x0,y0,fun);
loop:while(nextstep!=0){
flatten:switch(nextstep){
case 1:Block 1;nextstep=E^-2(A,x0,y0,fun);break;
case 2:Block 2;nextstep=E^-3(A,x0,y0,fun);break;
…
case 「n-1:Block n-1;nextsep=E^-n(A,x0,y0,fun);break;
case n:Block n;nextstep=E^-0(A,x0,y0,fun);break;
}}
```

图 5　基于全局代码的改进版压扁控制流模型

```
Initial:nexisiep=E^-1(A,x0,y0,fun);
loop:while(nextstep!=0){
flatten:switch(nextstep){
case 1:if(condition) nextstep=E^-2(A,x0,y0,fun);
else 「nextstep=E^-0(A,x0,y0,fun);break;
case 「2:Block1;nextstep=E^-3(A,x0,y0,fun);break;
case 3:Block 2;nextstep=E^-4(A,x0,y0,fun);break;
…
case 「n:Block n-1;nextsep=E^-n+1(A,x0,y0,fun);break;
case 「n+1:Block n;nextstep=E^-1(A,x0,y0,fun);break;
}}
```

图 6　基于循环基本块的改进版压扁控制流模型

本文在具体的算法实现过程中使用了以下的优化方法：

• 将混沌不透明谓词产生的过程封装在函数中，并将函数的实参存放在全局数组中，在混淆的过程中再通过调用实现。这样做的理由是：Horwitz 提出如果函数的参数中使用了指针或引用，潜在别名问题的数量将以指数形式增长[14]；而 Udupa 也提出使用全局数组能有效地搅乱静态分析器，提高程序的安全性[15]。

• 针对循环内的压扁控制流实现，我们可以抽出所有的循环基本块，结合经过独立混淆后的压扁块，在程序运行时随机选择源循环块或混淆后的基本块执行。这可以在一定程度上混淆保护者的意图。

3.3　实现框架

本文在实现压扁控制流及其改进算法的过程中，根据 JavaScript 的语法规则自定义文法文件 JavaScript. g，借助分析工具 Antlr3. 4 来自动生成词法和语法分析机制，并通过对基本文法文件的扩展 JavaScript_Ex. g，设计重要信息收集器来提取算法所需的内容。具体的实现框架应该是从源代码的输入直至混淆后代码输出的全过程，本文设计的框架如图 7 所示。

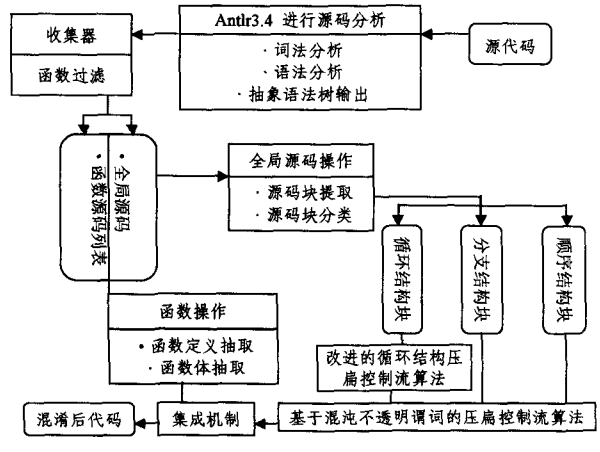

图 7　基于改进压扁控制流混淆的实现框架

4 实验结果及其分析

基于以上算法框架，借助 C# 语言的 VS2010 集成开发环境平台，本文开发了一个基于二维混沌 N 态不透明谓词的压扁控制流算法的 JavaScript 脚本语言混淆系统。使用 Google 公司开发提供的基准测试套件 V8 Benchmark Suite version7 作为测试用例，将以上改进的压扁控制流算法应用于这些用例中，并借助代码复杂度统计工具 Cyclomatic Complexity Analyzer(CCM)[16] 和 Google Chrome 开发者工具对混淆前后的代码进行静态分析和动态分析。静态分析包括代码行数、语句块数以及总圈复杂度；动态分析包括运行时间和占用内存。圈复杂度是衡量程序复杂度的度量指标[17]，CCM 能够统计 js 文件中每个函数的圈复杂度。通过多次实验，具体结果如表 1 所列。

表 1 混淆前后静态分析结果

测试例子	程序容量		代码行数		总圈复杂度	
	混淆前	混淆后	混淆前	混淆后	混淆前	混淆后
splay	11kB	13kB	394	469	51	95
crypto	47kB	53kB	1698	1905	265	428
raytrace	28kB	31kB	904	1157	79	217
regexp	109kB	121kB	1765	2631	185	335
earley-boyer	191kB	201kB	4684	5202	297	567

根据表 1 中的数据，从脚本大小的角度来看，混淆后程序容量和代码行数都明显增多，提高了代码复杂度。圈复杂度则能反映代码结构的复杂性，5 个基准测试例子的总圈复杂度混淆前后增加了 86.27%、67.19%、90.91%、174.68% 和 81.08%。

表 2 混淆前后 Google Chrome 动态性能分析输出结果

测试例子	运行时间		占用内存	
	混淆前	混淆后	混淆前	混淆后
splay	1ms	1ms	12.32MB	35.86MB
crypto	5ms	6ms	11.97MB	34.26MB
raytrace	2ms	3ms	7.89MB	34.45MB
regexp	5ms	6ms	12.31MB	34.73MB
earley-boyer	14ms	16ms	12.30MB	34.60MB

根据表 2 Google Chrome 的动态分析结果可知：

- 分析结果的正确输出表明了混淆后代码的正确性。
- 混淆后代码的性能有所下降。基准测试例子 splay、crypto、raytrace、regexp、earley-boyer 运行的 CPU 时间增幅分别为：0%、20%、33%、20%、14%，所占内存的增幅分别为：191%、186%、337%、182%、181%。

同时，本算法还能有效抵制“使用-定值量”的常量逆向传播分析。常量传播分析是基于对程序 *nextstep* 变量进行数据流分析跟踪实现的，使用不透明谓词能够提高逆向分析的难度。图 8 展示了测试用例 splay.js 中的典型函数使用改进前后控制流压扁算法的变形后代码，其中方框中内容表示构造混沌不透明谓词算法的实现方法，其参数 ValueIsOne 表示结果恒为 1 的秘钥，其他类推。相比左边的简单常数，基于混沌不透明谓词的压扁控制流算法实现代码具有更高的安全性能，通过不透明谓词来确定 *nextstep* 取值能够有效复杂化数据流程，并借助混沌算法的随机性来构造不透明谓词，使得逆向分析者无法简单确定 *nextstep* 在各 case 语句块的取值，达到提高代码抵制逆向分析的能力。

```
SplayTree.Nde.prototype.traverse_=
function(f){
var current = this;
var next = 1;
while(next!=0)
 {
  switch(next)
   {
 case 1: if(current)next=2;
 else next=0;break;
 case 2:var left=current.left;
 next=3;break;
 case 3:if(left){ left.traverse_(f);};
 next=4;break;
 case 4:if(current);next=0;break;
 case 5:current=current.right;
 next=1;break;
 }}}
```

```
SplayTree.Nde.prototype.traverse_=
function(f){
var current = this;
var next = 1;
while(next!=0)
{
switch(next)
{
case ChaoOpprecite(ValueIsOne):
if(current)next=2;
else next=0;break;
case ChaoOpprecite(ValueIsTwo):
var left=current.left; next=3;break;
case ChaoOpprecite(ValueIsThree):
if(left){ left.traverse_(f);};
next=4;break;
case ChaoOpprecite(ValueIsFour):
if(current);next=0;break;
case ChaoOpprecite(ValueIsFive):
current=current.right;next=1;break;}}}
```

图 8 改进前后压扁控制流算法的应用实例

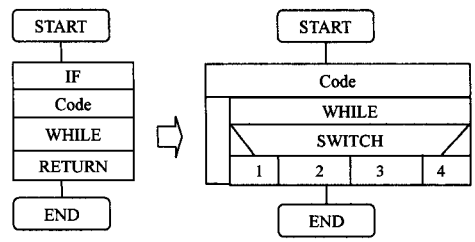

图 9 改进的压扁控制流算法实现前后代码 NS 图(全局顺序结构)

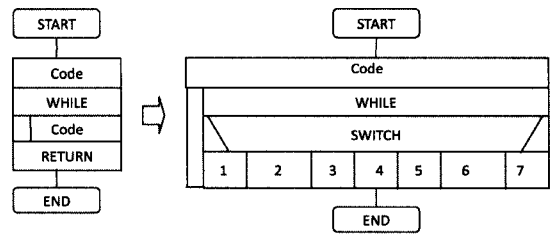

图 10 改进的压扁控制流算法实现前后代码 NS 图(while 循环结构)

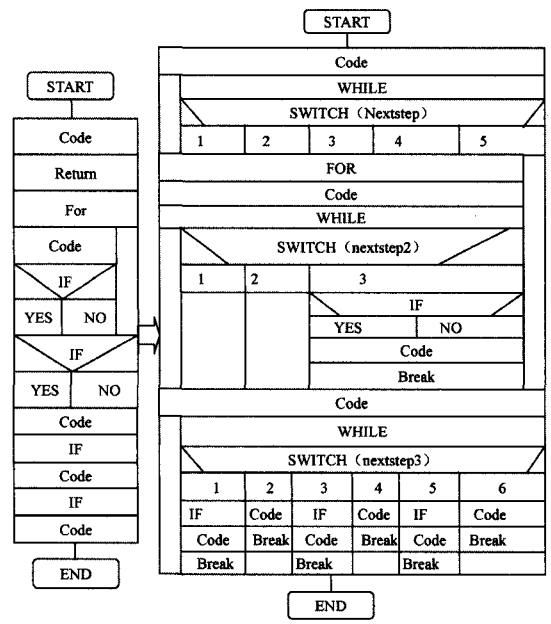

图 11 改进的压扁控制流算法实现前后代码 NS 图(综合版)

为了使混淆前后的代码流程结构更加明显，本文使用 CodeToFlowchart 流程图自动生成软件进行代码结构可视化

展示。选择 splay.js 中较为典型的函数进行测试，证实了经过压扁控制流算法混淆之后能有效地改变原先代码的流程结构，使得控制流结构更加复杂，增加破解者进行逆向分析的难度。具体结果如图 9—图 11 所示。

结束语 本文使用 Arnold cat 二维混沌映射函数来构造 N 态不透明谓词，提出并实现了基于混沌不透明谓词的压扁控制流算法，该算法是实现代码混淆的重要方法。通过开发一个基于此算法的 JavaScript 脚本混淆系统，并使用 Google 公司开发的基准测试套件 V8 Benchmark Suite version7 作为测试用例进行实验验证，借助圈复杂度统计工具 CCM 以及 Google Chrome 开发者工具对混淆前后的代码进行静动态分析，证实了此混淆算法的正确性和有效性。然而，代码混淆的程度提高与性能往往是相互矛盾的，通过更加复杂地混淆能提高逆向分析的难度，但同时也降低了程序性能。因此，如何平衡混淆程度和代码性能将是以后的研究方向。

参 考 文 献

[1] Collberg C, Thomborson J, Low D. A Taxonomy of Obfuscating Transformations[R]. Department of Computer Science, The University of Auckland, 1997

[2] Wang Chen-xi. A security architecture for survivability mechanisms[D]. Charlottesville: University of Virginia, 2001

[3] Borello J M, Mé L. Code Obfuscation Techniques for Metamorphic Viruses[J]. Journal of Computer Virology, 2008, 4(3): 211-220

[4] Weaver N, Paxson V, Staniford S, et al. A Taxonomy of Computer Worms[C]// Proceedings of the 2003 ACM workshop on Rapid malcode, Washington DC, USA, 2003. ACM Press, 2003: 11-18

[5] Collberg C, Nagra J. Surreptitious Software Obfuscation, Watermarking, and Tamperproofing for Software Protection[M]. Beijing: POST&TELECOM PRESS, 2012: 188-191

[6] 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

[7] Arboit G. A method for watermarking java programs via opaque predicates[C]// The Fifth International Conference on Electronic Commerce Research (ICECR-5). 2002: 102-110

[8] 苏庆，吴伟民，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)：155-160

[9] Matthews R. On the derivation of a "chaotic" encryption algorithm[J]. Cryptologia, 1989, 13(1): 29-42

[10] Dawei Z, Guanrong C, Wenbo L. A chaos-based robust wavelet-domain watermarking algorithm[J]. Chaos, Solitons & Fractals, 2004, 22(1): 47-54

[11] 张健，于晓洋，任洪娥，等. 一种改进的 Arnold Cat 变换图像置乱算法[J]. 计算机工程与应用，2009，45(35)：14-17

[12] Chow S, Gu Y, Johnson H, et al. An Approach to the Obfuscation of Control-flow of Sequential Computer Programs[C]// Proceedings of the 4th International Conference on Information Security, Malaga, Spain, 2001. Springer-Verlag, 2001, 2200: 144-155

[13] László T, Kiss A. Obfuscating C++ Programs via Control Flow Flattening[C]// Proc. of the 10th Symposium on Programming Languages and Software Tools. 2007

[14] Horwitz S, Reps T, Binkley D. Interprocedural slicing using dependence graphs[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1990, 12(1): 26-60

[15] Udupa, Sharath K, Saumya K. Debray, and Matias Madou[C]// Deobfuscation: Reverse engineering obfuscated code. Reverse Engineering, 12th Working Conference on. IEEE, 2005

[16] Blanck J. cyclomatic complexity analyzer(CCM) [OL]. http://www.blunck.info/ccm.html. 2012

[17] 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

(上接第 159 页)

[7] Jiang C L, Wang M M, Shu F. Multi-user MIMO with limited feedback using alternating codebooks [J]. IEEE Transactions on Communications, 2012, 60(2): 333-338

[8] Wang Jiu-teng. Joint MMSE equalization and power control for MIMO system under multi-user interference [J]. IEEE Communications Letters, 2012, 16(1): 54-56

[9] Maddah-Ali M, Motahari A, Khandani A. Communication over MIMO X channels: Interference alignment, decomposition, and performance analysis [J]. IEEE Transactions on Information Theory, 2008, 54(8): 3457-3470

[10] Rao Xiong-bin, Ruan Liang-zhong, Lan K N. Csi feedback reduction for mimo interference alignment[J]. IEEE Transactions on Signal Processing, 2013, 61(18): 4428-4437

[11] 田心记，倪水平. MIMO-MAC 中改进的空时码传输方案[J]. 北京邮电大学学报，2013，36(4)：95-98

[12] Li Liang-bin, Jafarkhani H, Jafar S A. When alamouti codes meet interference alignment: transmission schemes for two-user x channel [C]// IEEE International Symposium on Information Theory. Saint-Petersburg, 2011: 2577-2581

[13] Long Shi, Zhang Wei, Xia Xiang-gen. Space-Time Block Code Designs for Two-User MIMO X Channels[J]. IEEE Transactions on Communications, 2013, 61(9): 3806-3815

[14] Zaki A, Wang Chao, Rasmusse L K. Combining interference alignment and alamouti codes for the 3-user mimo interference channel[C]// IEEE Wireless Communications and Networking Conference. Shanghai, 2013: 3563-3567

[15] Ganesan A, Srajan B. Interference alignment with diversity for the 2×2 X network with three antennas [OL]. http://arxiv.org/abs/1304.1432. 2013.4

[16] Li Feng, Jafarkhani H. Space-time processing for x channels using precoders [J]. IEEE Transactions on Signal Processing, 2012, 60(4): 1849-1861