IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

DOI:10.1587/transfun.2024EAL2035

Publicized:2024/12/16

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

LETTER

An N-state Opaque Predicate Obfuscation Algorithm Based on Henon Map

Yindong CHEN[†*a)], ***Member***, **Wandong CHEN**[†], ***and*** **Dancheng HUANG**[†,††], ***Nonmembers***

SUMMARY In recent years, chaos maps and opaque predicates have received widespread attention in the field of code obfuscation. Chaos map is proved opaque in n-state predicates code obfuscation. We use n-state opaque predicates to improve the control flow flattening technique and flatten the control structure of the code. Finally, we demonstrate that Henon map scheme outperforms other obfuscation schemes.

key words: *chaos map, opaque predicate, code obfuscation*

1. Introduction

Chaos map defines the nonlinear relationship between system states and time, and these states exhibit complex and sensitive dynamic behaviors. That is, small changes in the initial conditions can cause huge differences in the evolution of the system [1]. It is especially useful in computer science and cryptography because it has a high degree of randomness and unpredictability. In the field of software obfuscation, they can be used to build opaque predicates. These properties of chaos map can be exploited to improve the effectiveness of opaque predicates. Applying chaos map to the design of opaque predicates not only expands the method of constructing opaque predicates, but also improves the obfuscation strength, uncertainty, security and complexity of obfuscated code. We use Henon map to improve the traditional control flow obfuscation algorithm, and experiments have proven that the effect is effective.

In the process of exploring code protection mechanisms, code obfuscation techniques were classified for the first time and innovatively subdivided into four main types: layout obfuscation, data obfuscation, control flow obfuscation, and preventive obfuscation [3]. Wang et al [4] proposed a control flow flattening algorithm. This algorithm is a code obfuscation technique used to change the logical structure of the program, making the control flow graph flatter and hindering reverse engineering. Arboit [5] proposed to use the quadratic remainder theory to construct opaque predicates and combine it with watermarking technology. However, later studies found that this method performed poorly in terms of safety. Su et al [6] proposed a method to construct opaque predicate clusters through improved Logistic chaos map. Through experimental verification, they proved the feasibility of two-dimensional obfuscation map, and the method has strong security features when the result is true. Wang et al [2] proposed a flattened control flow method based on congruence equations and improvements, which can effectively resist dynamic attacks.

The structure of the letter is as follows. Sect.2 introduces some preliminaries on chaos map and opaque predicate as well as control flow algorithms. In Sect.3, we propose a method to construct opaque predicates using Henon map, and than use the constructed chaotic opaque predicate to improve the control flow flattening algorithm. In Sect.4, Experiments and analysis are introduced. Finally, Sect.5 concludes the letter.

2. Preliminaries

2.1 Chaos

Definition 1. *(Chaos [7]) A continuous self-mapping $F(n)$ on the interval I has chaotic properties if it meets the following conditions:*
i) Mapping periodic points on F has no upper bound;
ii) Given an uncountable subset N on the closed interval I, for $\forall m, n \in N, m \neq n$

- $\lim_{n\to\infty} \sup \left|F''(n) - F''(m)\right| > 0$;
- $\lim_{n\to\infty} \left|F''(n) - F''(m)\right| = 0$;
- $\lim_{n\to\infty} \sup \left|F''(n) - F''(m)\right| > 0$, *$m$ is a periodic point of F.*

2.2 Code obfuscation

Code obfuscation can be expressed by the following for-

[†]The author is with the Department of Computer Science and Technology, Shantou University, and the Key Laboratory of Intelligent Manufacturing Technology, Ministry of Education, China.

[††]The author is with the Department of Mathematics and Computer Science, Guangdong Technion-Israel Institute of Technology, China.

[*]The work is supported by the Key Research Platforms and Projects of Higher Education Institutions in Guangdong Province (No. 2024ZDZX1021, 2024KSYS012), the Natural Science Foundation of Guangdong Province (No. 2514050003605), and the Science and Technology Planning Projects of Shantou (No. 220516096491783).

a) E-mail: ydchen@stu.edu.cn

mula: $\mathrm{Obf}(P) = P'$, where Obf is an obfuscation algorithm, P is the original program, and P' is a program that is functionally equivalent to P after obfuscation.

$$\forall I, \quad \mathrm{Exec}(P, I) = \mathrm{Exec}(P', I),$$

where, I represents the input of the program, and Exec is the function executed by the program, which executes program P or P' under the given input I. This shows that for any input I, the execution results of the original program P and the obfuscated program P' are the same, thus ensuring that obfuscation will not change the function of the program. According to the conversion target and logical structure of the original program, code obfuscation technology is divided into layout obfuscation, data obfuscation, preventive obfuscation, and control flow obfuscation [3].

2.3 Opaque Predicate

Opaque predicate falls under the category of control flow obfuscation. It is an expressions that always evaluate to true or false at compile time or run time, but whose results are difficult to predict without global knowledge or deep understanding of the specific runtime context and program state. Logical formulas for opaque predicate can be expressed as follows.

$$P(x): \quad X \to \{\mathrm{true}, \mathrm{false}\},$$

where P is a predicate function, X is a set of inputs, and $P(x)$ appears to be true or false to the analyst expression, but in practice it always returns the same result value, regardless of the value of x. If $\forall x \in X, P(x) = \mathrm{true}$ is always holds, then P is defined as a truth opaque predicate, denoted P^{T}; If $\forall x \in X, P(x) = \mathrm{false}$ is always holds, then P is defined as the never false opaque predicate, denoted P^{F}.

Trapdoor opaque predicates. Let $j \in \{1, 2, \cdots, n\}$ and K_j denote the key of some constructed predicate. If K_j is known before the program execution, knowing K_j allows the analyst to determine the result of the predicate execution. Conversely, not knowing K_j makes it difficult or impossible to determine the result.

N-state opaque predicate. For $\forall o \in O$, opaque predicates expression $P = E(o) \in Z$, where $Z = \{0, 1, 2, \cdots, n\}$, produces the result not limited to true or false, but maps it to the natural numbers $1, 2, ..., n$. The P corresponding to the mapping E forms an n-state opaque predicate.

2.4 Control Flow Flattening

Control flow flattening is a control flow obfuscation technique, aimed at altering the original control flow structure of a program to make its logic difficult to understand and analyze. This technique hides the original structure of basic blocks and control transitions by transforming the control flow of the program into a single scheduling structure, typically a large "switch-case" statement. The core idea of control flow flattening is to reconstruct the multiple branch structures in the program (such as "if-else", "for", "while", etc.) into seemingly disordered state transition processes controlled by a unified dispatcher. The transformed structure retains the same functionality as the original program logically, but hides the real execution path of the program in form, which increases the difficulty of attack analysis.

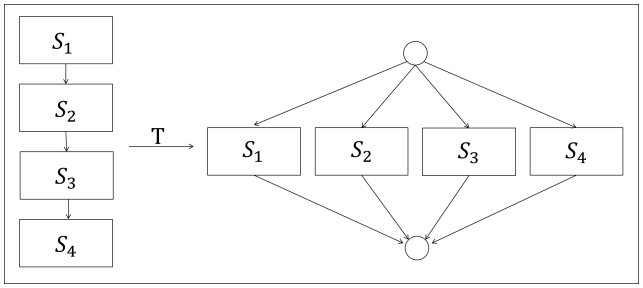

Fig. 1 Control Flow Flattening

3. Opaque Predicate Methods Based on Henon Map

3.1 Henon Map

Henon map is a two-dimensional chaotic system that generates real number sequences with characteristics that depend on initial parameters and are pseudo-random behavior and limited range of truth values. The mathematical form of the expression of this two-dimensional chaotic map is as follows.

$$\begin{cases} X_{n+1} = Y_n + 1 - \alpha X_n^2 \\ Y_{n+1} = \beta X_n \end{cases} \qquad \begin{pmatrix} -1.5 < X < 1.5 \\ -0.4 < Y < 0.4 \end{pmatrix}$$

When $1.07 \leq \alpha \leq 1.4$ and $\beta = 0.3$, the system is in a chaotic state. When $\alpha = 1.4$, the system complexity is maximum.

3.2 Characteristics of Henon Map

Firstly, Henon map is highly sensitive to initial conditions, which is beneficial to increase the uncertainty of the whole confusing system. Meanwhile, note that the iterative functions with different parameters all show difficult to predict pseudo-random phenomena. Henon map has good pseudo-random characteristics, and chaotic sequences show long-term unpredictability and high complexity, which further improves the security of chaotic systems. Moreover, Henon map is a discrete chaotic mapping with larger parameter and initial value range, larger key space of chaotic system, and

uniform distribution of chaotic sequence in state space. This property is beneficial to generate uniform state sequences covering the global space, and it is convenient to evenly partition and map state spaces into real integer sequences to improve the construction efficiency of n-state chaotic opaque expressions.

Therefore, due to the above characteristics, Henon map is used in opaque predicates construction and is expected to show excellent security performance.

3.3 Opaque Predicate Methods

In this subsection, an n-state chaotic opaque predicate generated by Henon chaotic map is proposed. This new predicate can be applied to the enhanced flat control flow algorithm to improve the program's ability to resist reverse engineering. The following is the construction of n-state opaque predicate based on two-dimensional chaotic Henon map.

Algorithm 1: N-state Opaque Predicate Based on Henon Map

1. Generate a set of n integers by random function to form $Z^* = \{Z_1^*, Z_2^*, \ldots, Z_n^*\}$.
2. Generate random real two-dimensional vector sequence $M = \{M_0, M_1, \cdots, M_n\}$ using initial quadruple key $(\alpha, \beta, X_0, Y_0)$ with Henon chaotic map, where $M_0 = [X_0, Y_0]'$, and so on.
3. Map a sequence of real two-dimensional vectors $M = \{M_0, M_1, \cdots, M_n\}$ to a sequence of real integers by mapping Fun.
   $M = \{M_0, M_1, \cdots, M_n\} \xrightarrow{\text{Fun}} N = \{N_0, N_1, \cdots, N_n\}$. Let $\forall Z^* \in [-m, m]$ and adopt the mapping function as shown in equation. For x_n, y_n : $X_n = \frac{x_n - \min(x)}{\max(x) - \min(x)} - \frac{1}{2}$, $Y_n = \frac{y_n - \min(y)}{\max(y) - \min(y)} - \frac{1}{2}$, $N_i = \text{Round}\{[X_n + Y_n] \times m\}$, where $M(X_i), M(Y_i)$ represent the data of the two-dimensional matrix M, and Round is the rounding function. The key becomes the five-tuple $(\alpha, \beta, X_0, Y_0, Fun)$.
4. Then let $result$ be the same amount of storage space and find $\left(N \xrightarrow{\text{mapping}} Z^*\right) \rightarrow result$, $0 \leq result \leq n$, The count of the number of elements in N that are the same as z^* is stored in $result$.
5. Repeat Step 2 to Step 4 to train keys that can produce different result values until each value has a corresponding key. Might as well make it as $\{result^{=1}, result^{=2}, \cdots, result^{=n}\}$, where $result^{=i}$ denotes the one with constant key of $i \in \{1, 2, \cdots, n\}$.

Let's take a simple "while" loop for example. The loop condition, loop body, and conditional branches can all be visualized in a control flow graph. However, with the help of control flow flattening techniques, the original explicit process structure can be rewritten. During flattening, the existing nested loops and conditional branch statements are merged into a single "switch" structure and placed inside a "while" loop that never exits. Each basic block of the control flow graph is

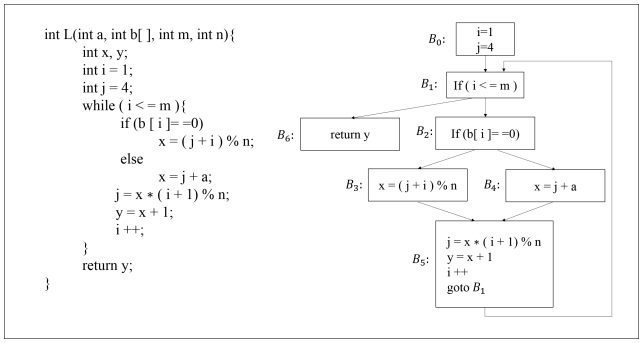

Fig. 2 Code Before Obfuscation

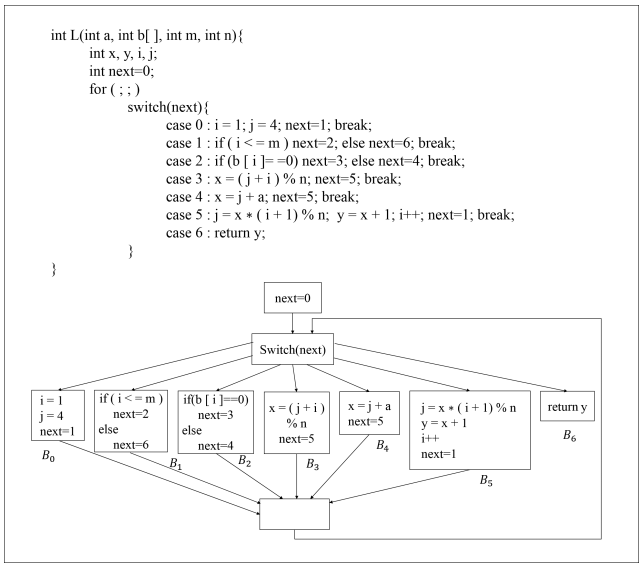

Fig. 3 Control Flow Transformation Code

translated into a "case" statement in a "switch" structure.

This algorithm maintains the correct control flow structure by updating the value of the next variable in each basic block continuously. Even if the structure of the control flow graph is disturbed, the runtime control flow still passes through the basic blocks in a correct order. Because each basic block loses the jumping point information, the cracker can only gradually track basic blocks that have been executed, which greatly increases the analyzing difficulty.

The attacker can deduce the execution flow of the program and reconstruct the control flow graph by using the "use-define" chain constant propagation analysis on the next variable of the "switch" statement.

To prevent this analysis, the data flow analyzer must be further obfuscated to keep track of the value of the next variable. If the data flow analyzer cannot determine the value of the next variable, the attacker also cannot determine the specific path of the control flow, so that the control flow graph cannot be reconstructed accurately. In this letter, the n-state opaque

```
int L(int a, int b[ ], int m, int n){
        int x, y, i, j;
        int next=E^=0 (α, β, X0, Y0, Fun);
        for ( ; ; )
                switch(next){
                        case chaos(value0) : i = 1; j = 4; next= E^=1 (α, β, X0, Y0, Fun); break;
                        case chaos(value1) : if ( i < = m ) next= E^=2 (α, β, X0, Y0, Fun); else next= E^=6
(α, β, X0, Y0, Fun); break;
                        case chaos(value2) : if (b [ i ]= =0) next= E^=3 (α, β, X0, Y0, Fun); else next= E^=4
(α, β, X0, Y0, Fun); break;
                        case chaos(value3) : x = ( j + i ) % n; next= E^=5 (α, β, X0, Y0, Fun); break;
                        case chaos(value4) : x = j + a; next= E^=5 (α, β, X0, Y0, Fun); break;
                        case chaos(value5) : j = x * ( i + 1) % n;  y = x + 1; i++; next= E^=1(α, β, X0, Y0, Fun); break;
                        case chaos(value6) : return y;
                }
}
```

Fig. 4 Improved Control Flow Flattening

expression is used to calculate the value of the variable next. Chaos is the obfuscating function to obfuscate the constant term.

4. Experiment and Analysis

In order to make the structure of the code flow before and after obfuscation clearer, we use five classical algorithms as test cases, and the tests are the cyclomatic complexity of the function, Non-comment Lines of Code (NLOC), and the number of tokens confused. Our algorithm is compared with the algorithm of Ref. [2] and [6]. "Original" is the source code metric before obfuscation, followed by the results after obfuscation.

Table 1 Non-comment Lines of Code

NLOC	Original	Su	Wang	Ours
MergeSort	73	183	298	376
HeapSort	69	174	224	328
Dijkstra	74	249	368	469
Primd	84	268	372	435
KMP	62	193	271	392

Table 1 is the comparison of the number of code lines before and after obfuscation. It can be seen that the effect of obfuscation is different for different codes, mainly due to the different control flow after code obfuscation. Table 1 shows the obfuscation results of different algorithms. Su's algorithm is about 2 to 3 times, Wang's about 3 to 4 times, and ours about 5 to 6 times.

Table 2 Cyclomatic Complexity Number

CCN	Original	Su	Wang	Ours
MergeSort	9	14	19	25
HeapSort	12	20	26	34
Dijkstra	11	19	28	47
Primd	13	18	24	32
KMP	12	22	27	42

Table 2 shows the obfuscation effect for each different code. It can be observed that the obfuscated code not only increases in the non-comment lines of code, but also significantly improves the cyclomatic complexity, which indicates the complexity of the code structure. For the five benchmarks, their total cyclomatic complexity increases by 177%, 183%, 327%, 146%, and 250%, which are better than those of other algorithms.

Table 3 Confused Token Number

Token	Original	Su	Wang	Ours
MergeSort	345	731	851	1276
HeapSort	260	634	712	1194
Dijkstra	422	752	943	1645
Primd	486	1043	1279	2036
KMP	421	947	1154	1964

Table 3 compares the number of confused tokens (tokens represent symbols such as keywords, operators, identifiers, and separators). The number of tokens reflects part of the complexity of the code: the greater the number of tokens, the higher the total complexity of the code. Table 3 shows that our results have significant advantages over others.

5. Conclusion

Based on the two-dimensional chaotic model of Henon map, an n-state opaque predicate is constructed to enhance the randomness and uncertainty of the obfuscation scheme. It significantly increases the cyclomatic complexity of the code. Therefore, it further increases the difficulty of code reverse analysis.

References

[1] Xie X., Liu F., Lu B. and Xiang F., Mixed Obfuscation of Overlapping Instruction and Self-Modify Code Based on Hyper-Chaotic Opaque Predicates[C]. The 10th International Conference on Computational Intelligence and Security, Kunming, China, 2014, 524-528.

[2] Wang Y., Huang Z., Gu N.. Obfuscating algorithm based on congruence equation and improved flat control flow[J]. Journal of Computer Applications, 2017, 37(6): 1803-1807 (in Chinese).

[3] Collberg C., Thomborson C., Low D.. Manufacturing cheap, resilient, and stealthy opaque constructs[C]. Proc. of the ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, 1997, 184-196.

[4] Wang C., Advisor J., Blakley G., et al. A Security Architecture for Survivability Mechanisms[D]. University of Virginia. 2001.

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

[6] Su Q., Wu W., Li Z.. Research and application of chaos opaque predicate in code obfuscation[J]. Computer Science, 2013, 40(6): 155-159 (in Chinese).

[7] Li T., Yorke J.. Period Three Implies Chaos[J]. The American Mathematical Monthly, 1975, 82(10): 985-992.