Obfuscate Arrays by Homomorphic Functions

William Zhu, *Student Member, IEEE,* Clark Thomborson, *Senior Member, IEEE,*
and Fei-Yue Wang, *Fellow, IEEE*

***Abstract*— As various computers are connected into a world wide network, software protections becomes a more and more important issue for software users and developers. There are some technical measures for software protections, such as hardware-based protections and software-based techniques, etc. Software obfuscation is one of these measures. It protects software from unauthorized modification by making software more obscure so that it is hard for the potential attacker to understand the obfuscated software. Chow et al. use residue number technique to software obfuscation by encoding variables in the original program to hide the true meaning of these variables [1]. There is some discussion about the division of residue numbers in [1], but, in order to lay a sound ground for this technique, we proposed homomorphic functions in [2] to deal with division by several constants in residue numbers.**

Data structures are important components of programme and they are key clues for people to understand codes. Obfuscating data structures of programme will make it very hard for an enemy to attack them. In this paper, we apply homomorphic functions to obfuscating the data structures of software.

***Index Terms*— software obfuscation, residue number coding, homomorphic obfuscation.**

I. Introduction

As the Internet develops rapidly and becomes popular, software security appears a vital issue for computer industry and even for our society. The demand for software protections is stronger and stronger. To achieve this goal, researchers has proposed various techniques, such as dongle, a hardware-based method, and software watermarking, a software-based approach.

Software obfuscation [3], [4], [5] is a measure for software security. It transforms a program into a new one that is harder to understand than the original one. In software obfuscation, variable transformation is a major method to transform software into a new one that is hard for attackers to understand.

Chow el at. applied the residue number coding [6], an approach used in hardware design, high precision integer arithmetic, and cryptography, to software obfuscation [1], [7]. In a previous paper [2], we proposed the concepts of homomorphic functions, a potential area for further exploration. Based on these concepts, we established a sound ground for residue number coding for software obfuscation. Especially, we used this to develop an algorithm for division by several constants in order to strengthen some points in an earlier publication [1]. In this paper, we describe further applications of homomorphic functions to software obfuscation.

The remainder of this paper is structured as follows. Section II describes the backgrounds and techniques of software obfuscation. Section III is the focus of this paper. It applies homomorphic functions to array transformations. This paper concludes in section IV.

William Zhu is with the Department of Computer Sciences, The University of Auckland, Auckland, New Zealand and the Institute of Automation, The Chinese Academy of Sciences, Beijing 100080, China. (corresponding author to provide phone: +64-9-3737-599ext.82289; fax: +64-9-3737-453; e-mail: fzhu009@ec.auckland.ac.nz)

Clark Thomborson is with the Department of Computer Sciences, The University of Auckland, Auckland, New Zealand. e-mail:cthombor@cs.auckland.ac.nz

Fei-Yue Wang is with the Institute of Automation, The Chinese Academy of Sciences, Beijing 100080, China and the Systems and Industrial Engineering Department, The University of Arizona, Tucson, AZ 85721, USA. e-mail: feiyue@sie.arizona.edu

The first two authors are in part supported by the New Economy Research Fund of New Zealand and the first and the third author is in part supported by the National Outstanding Youth Research Programme of China

II. Backgrounds

A. Software protection

As the Internet evolves rapidly, software piracy is rampant in the world, as a result, software protection becomes a vital issue in current computer industry and a hot research topic [1], [3], [4], [5], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17].

There are legal measures and technical approaches for software protection. Legal protection has become increasingly important since more software is distributed without a signed license agreement. Legal measures are copyright, patent, registration and license. Software copyright protects the exclusive rights of a software developer to reproduce or copy, adapt, distribute and publicly perform the work. Generally, copyright laws protect the form of expression of an idea, but not the idea itself. With respect to software, this typically means that it protects a computer program but not the methods and algorithms within the program. Thus, source code and object code are protected against literal copying. While software copyright protects only the expression of an idea in software, software patent protects the underlying idea and features of software. Even independent reinvention of the same technique by others does not give them the right to use it. The protection of software by registration has long been accepted internationally. In most countries, rights are initially secured by registration and maintained by later use in the country. A software license is a contract between a producer and a user of computer software. It gives the user the privilege to use software in accordance with the conditions of the license. That privilege may be revoked by the producer at any time, with or without cause.

While legal protection in a country generally can not be extended to other nations and obtaining patent protection for software is relatively expensive, software vendors still seek technical measures to protect their software. Technical approaches can be classified as hardware-based methods and software-based methods.

1-4244-0134-8/06/$20.00 © 2006 IEEE

A dongle is a typical hardware-based method. It is a small hardware device that plugs into the serial or USB port of a computer to ensure that only authorized users can use certain software applications. They are only used with expensive, high-end software programs. When a program that comes with a dongle runs, it checks the dongle for verification as it is loading. If it does not find the dongle, the computer quits.

Currently, this is the most reliable method of protecting software and is also a convenient approach of prevent commercial software of high price, such as accounting and inventory management applications, legal and corporate systems, building construction estimates, CAD systems, etc. from piracy. But, a dongle is a physical device, thus it is not convenient for small software and for online distribution.

The current software-based methods are code authentication, server side execution, code encoding, software watermarking, software obfuscation, and etc. Code authentication is most efficient when authentication data are sent through network, but user has complete code, which in theory can be mangled, authentication procedures can be removed. For server side execution, software developer does not send final code to users, but provide users the service of the software through executing whole or part of the software on a remote server. It can be used only in presence of high availability of broadband networks. Code encoding protects against tampering of program; in present programs used very often; main drawback is that decoder can be written and used as a universal tool. Software watermarking tries to insert a secret message, called a software watermark, into software as the evidence of ownership of it [18]. Software obfuscation translates software into a semantically-equivalent one that is hard for attacker to analyze. This paper will focus on software obfuscation.

B. Software obfuscation

Since Sun Microsystems designed Java, a programming language, in the mid-1990s, it is widely used to deliver interactive web content in the Internet, such as video displays, animations, and interactive games. As the Internet is connected with various heterogeneous hardware with different architectures, Java is intended to be architecture-independent.

Traditionally, to develop software, we first write source code for the software in some high level programming language, then compile the source code to get a program, called native code, that will execute only on a specific type of CPU with a specific operating system.

As for Java, the finished product of software, bytecode, is actually not the code in the architecture-specific machine language which can be understood by any computers, but something that can be run on a "Java virtual machine" (JVM), a platform between the high-level language and the real computer. It is the JVM that makes the distributed executables of Java programs portable, not architecture-dependent.

As we know, Java bytecode is architecture-independent, thus it contains a lot of information of the source codes and that make it easy to decompile the source codes and extract important algorithms from them. The feature of Java bytecode helps to reverse-engineer a Java executable and results in the software piracy, unauthorized penetration and system modification.

After mobile code becomes popular at the Internet age, another type of software piracy occurs. As we know, mobile code migrates across a network from a remote source to a local system and is then run on that local system, which is a personal computer, or a mobile phone, or Internet appliance, so the software developers and owners may encounter piracy from malicious hosts.

In order to protect software, especially that in forms such as Java bytecode and mobile code, several measures has been proposed, among them software obfuscation seems a last approach. It transforms a program into another semantically equivalent one that is more hard to understand and reverse engineer. Paper [3], [4], [5], [10] had a detailed description of software obfuscation.

C. Array transformations

There are several algorithms of software obfuscation such as layout transformation, computation transformation, ordering transformation, data transformation [3]. Array index change, array folding, and array flattening are some of data transformations. As said in [3], by adding the data complexity in the program, array index change, array folding and flattening can make a program much more difficult to understand and reverse engineer.

1) Array index change: The following Fig. 1 is a simple example of array index change method [3].

```
int A[9];                    int A1[5], A2[4];
A[i] = ...                   if((i%2 == 0))
                      ⇒           A1[i/2] =...
                             else
                                 A2[i/2] = ...
```

Fig. 1. An example of array index change

2) Array folding and flattening: While array folding increases the dimension of an array in the code, such as transforming a 2-dimensional array to a 1-dimensional array, array flattening decreases the dimension of an array in the code, such as transforming a 2-dimensional array to a 4-dimensional array.

D. Homomorphic function

In one of our previous papers [2], we proposed the concept of homomorphic obfuscations, and developed an algorithm for division by several constants. We describe some basic concepts about residue number coding as follows. Details of them can be found in [2].

Let Z be the set of all integers, n a given positive integer. For any $x \in Z$, denote $[x]_n = \{y \in Z \mid y - x \text{ is divisible by } n\}$ and we call $[x]_n$ the residue class of x modulo n. We omit the subscript when there is no confusion. Let Z/nZ be the set of all these residue classes with respect to modulo n, where $Z/nZ = \{[0], [1], \ldots, [n-1]\}$.

Z/nZ has three operations +, -, $\times$ defined as follows: for any two $[x], [y] \in Z/nZ$, $[x] + [y] = [x + y]$, $[x] - [y] = [x - y]$, $[x] \times [y] = [x \times y]$.

The product $Z/m_1Z \times Z/m_2Z \times \ldots \times Z/m_kZ$ also has three operations $+$, $-$, $\times$ defined as follows: for any two $([x_1]_{m_1}, \ldots, [x_k]_{m_k})$, $([y_1]_{m_1}, \ldots, [y_k]_{m_k}) \in Z/m_1Z \times \ldots \times Z/m_kZ$,

$$\begin{aligned}&([x_1]_{m_1}, \ldots, [x_k]_{m_k}) + ([y_1]_{m_1}, \ldots, [y_k]_{m_k})\\ &\quad = ([x_1 + y_1]_{m_1}, \ldots, [x_k + y_k]_{m_k})\end{aligned}$$

$$\begin{aligned}&([x_1]_{m_1}, \ldots, [x_k]_{m_k}) - ([y_1]_{m_1}, \ldots, [y_k]_{m_k})\\ &\quad = ([x_1 - y_1]_{m_1}, \ldots, [x_k - y_k]_{m_k})\end{aligned}$$

$$\begin{aligned}&([x_1]_{m_1}, \ldots, [x_k]_{m_k}) \times ([y_1]_{m_1}, \ldots, [y_k]_{m_k})\\ &\quad = ([x_1 \times y_1]_{m_1}, \ldots, [x_k \times y_k]_{m_k})\end{aligned}$$

III. Application of homomorphic function to array's dimension change

HOMOMORPHIC functions can be used to obfuscate programs through changing the index or the dimension of an array in the programs to obfuscate them. In the following, we describe in details four methods, which we reported in [19], to apply homomorphic functions to software obfuscation.

A. Index change

Homomorphic functions can be used to change the index of an array in software to obfuscate it. For an array A[n], the technique is as follows.

- Find an m such that $m > n$, and n and m are relatively prime.
- Change the array into another array B[n] and the element A[i] is turned into b[i*m mod n].

The homomorphic function f: $Z/nZ \rightarrow Z/nZ$ defined by $f([i]_n) = [i * m]_n$ is an isomorphism, thus the above replacement guarantees the semantics of the original program.

Example 1 (Index change): For the program in Fig. 2, we choose $m = 3$. The obfuscated program is in Fig. 3.

```
...
real A[100];
...;
S = 0;
for(i = 0; i < 100; i++)
  S = S + A[i];
...
```

Fig. 2. An unobfuscated program

```
...
real B[100];
...
S = 0;
for(i = 0; i < 100; i++)
    S = S + B[i*3 mod 100];
...
```

Fig. 3. An obfuscated version of the program in Fig. 2

B. Index and dimension change

Homomorphic functions can be used to change the index and the length of dimension of an array in programs to obfuscate them. For an array A[n], the following procedure can achieve this goal.

- Find an m such that $m > n$, and n and m are relatively prime.
- Change the array into another array B[m] and the element A[i] is turned into b[i*n mod m].

The above method can be regarded as two steps. Firstly, extend array A[n] to C[m] with C[i] = A[i] for $0 \leq i < n$ and C[i] undefined for $n \leq i < m$]. Then, change the array C[m] into b[m] by replacing C[i] with B[i*n mod m] as in the index change method.

Example 2 (Index and dimension change): For the program in Fig. 4, we choose $m = 101$. The obfuscated program is in Fig. 5.

```
...
real A[100];
...;
S = 0;
for(i = 0; i < 100; i++)
  S = S + A[i];
...
```

Fig. 4. An unobfuscated program

C. Array folding

Homomorphic functions can be used to increase the number of dimension of an array in software to obfuscate it. The technique is referred as to array folding. For an array A[n], we assume $n > 2$. The array folding procedure is as follows.

- If n is a prime, let $m = n + 1$; otherwise $m = n$.
- Extend A[n] into C[m] by C[i] = A[i] for $0 \leq i < n$ and C[i] undefined for $n \leq i < m$.
- Factor m into m_1 and m_2. Replace C[m] with B[m_1, m_2] through B[i mod m_1, i mod m_2] = C[i] for $0 \leq i < m$.
- Replace any A[i] with B[i mod m_1, i mod m_2] in the unobfuscated program.

Example 3 (Array folding): For the program in Fig. 6, we choose $m = 3$.

- Factor 100 into 4 $\times$ 25, two relatively prime integers.
- Turn the 1-dimension array A[100] into a 2-dimension array B[4, 25] and let B[i mod 4, i mod 25] = A[i] for $0 \leq i < 100$.

```
...
real B[101];
...
S = 0;
for(i = 0; i < 100; i++)
    S = S + B[i*100 mod 101];
...
```

Fig. 5. An obfuscated version of the program in Fig. 4

- Replace any A[i] with B[i mod 4, i mod 25] in the unobfuscated program.

The obfuscated program is in Fig. 7.

```
...
S = 0;
for(i = 0; i < 100; i++)
  S = S + A[i];
...
```

Fig. 6. An unobfuscated program

```
...
S = 0;
for(i = 0; i < 4; i++)
  for(j = 0; i < 25; j++)
    S = S + B[i, j];
...
```

Fig. 7. An obfuscated version of the program in Fig. 6

D. Array flattening

Homomorphic functions can be used to decrease the number of dimension of an array in software to obfuscate it. The technique is referred as to array flattening. For a two dimension array $A[n_1, n_2]$, the array flattening procedure is as follows.

- Find two relatively prime integers m_1 and m_2 such that $n_1 \leq m_1$ and $n_2 \leq m_2$. Let $m = m_1 * m_2$.
- Turn the 2-dimension array $A[n_1, n_2]$ into another 2-dimension array $C[m_1, m_2]$ by $C[i, j] = A[i, j]$ for $0 \leq i < m_1$ and $0 \leq j < m_2$, and $C[i, j]$ undefined otherwise. Replace all $A[i, j]$ with $C[i, j]$.
- Find two relatively integers k_1 and k_2 such that $k_1 * m_1 + k_2 * m_2 = 1$.
- Turn the 2-dimension array $C[n_1, n_2]$ into a 1-dimension array $B[m]$ and let $B[i] = C[i \ mod \ m_1, i \ mod \ m_2]$ for $0 \leq i < m$.
- Replace any $A[i, j]$ with $B[(i * k_1 + j * k_2) \ mod \ m]$ for $0 \leq i < n_1$ and $0 \leq j < n_2$ in the unobfuscated program.

Example 4 (Array flattening): For the program in Fig. 8,

```
...
S = 0;
for(i = 0; i < 4; i++)
  for(j = 0; i < 26; j++)
    S = S + B[i, j];
...
```

Fig. 8. An unobfuscated program

- We choose 4 and 27.
- Time 4 and 27 into 108
- Turn the 2-dimension array B[4, 25] into a 1-dimension array A[108] and let
  - A[i] = B[i mod 4, i mod 27] for $0 \leq i < 104$.
  - A[i] = any value for $104 \leq i < 108$.
- Replace any B[i mod 4, i mod 27] with A[i] for $0 \leq i < 104$ in the unobfuscated program.

The new obfuscated program is as in Fig. 9.

```
...
S = 0;
for(i = 0; i < 104; i++)
  S = S + A[i];
...
```

Fig. 9. An obfuscated version of the program in Fig. 8

IV. Conclusions

WHILE the residue number coding can be used in RSA cryptography, it also has applications to software obfuscation to encode variables to hide the real meaning of these variables [1]. This paper proposes applications of it to obfuscating data structures in software. We will investigate more applications of homomorphic functions to software obfuscation in our future research.

References

[1] Chow and et al, "Tamper resistant software encoding," *US patent*, vol. 6594761, pp. 1–32, Oct. 2003.

[2] W. Zhu and C. Thomborson, "A provable scheme for homomorphic obfuscationin in software security," in *The IASTED International Conference on Communication, Network and Information Security, CNIS'05*, Phoenix, USA, Nov 2005, pp. 208–212.

[3] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," in *Tech. Report, No.148, Dept. of Computer Sciences, Univ. of Auckland*, 1997.

[4] C. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation - tools for software protection," in *IEEE Transactions on Software Engineering*, vol. 28, Aug. 2002, pp. 735–746.

[5] S. Drape, "Obfuscation of abstract data-types," Ph.D. dissertation, The Universite of Oxford, 2004.

[6] D. Knuth, *The art of computer programming, Vol. 2, Seminumerical algorithms*, 3rd ed. Addison-Wesley, 1997.

[7] J. Nicherson, S. Chow, and H. Johnson, "Tamper resistant software: extending trust into a hostile environment," in *Proceedings of ACM Multimedia '01*. ACM Press, 2001.

[8] A. Majumdar and C. Thomborson, "On the use of opaque predicates in mobile agent code obfuscation," in *LNCS 3495*, May 2005, pp. 648–649.

[9] L. Ertaul and S. Venkatesh, "Novel obfuscation algorithms for software security," in *2005 International Conference on Software Engineering Research and Practice, SERP'05*, june 2005, pp. 209–215.

[10] G. Wroblewski, "A general method of program code obfuscation," Ph.D. dissertation, Wroclaw University, 2002.

[11] J. Ge and S. C. A. Tyagi, "Control flow based obfuscation," in *DRM'05*. ACM, Nov. 2005, pp. 83–92.

[12] M. D. Preda and R. Giacobazzi, "Semantic-based code obfuscation by abstract interpretation," in *Proceedings of the 32th Internarional Colloquium on Automata, Language and Programming*, vol. 3580. Springer Verlag, 2005, pp. 1325–1336.

[13] Y. Sakabe, M. Soshi, and A. Miyaji, "Java obfuscation approaches to construct tamper-resistant object-oriented programs," *IPSJ Digital Courier*, vol. 1, pp. 349–361, 2005.

[14] T. Toyofuku, T. Tabata, and K. Sakurai, "Program obfuscation scheme using random number to complicate control flow," in *EUC Workshops 2005*, ser. LNCS, vol. 2823, 2005, pp. 916–925.

[15] A. Monden, A. Monsifrot, and C. Thomborson, "A framework for obfuscated interpretation," in *Australia Information Security Workshop 2004*, 2004, pp. 7–16.

[16] M. Sosonkin, G. Naumovich, and N. Memon, "Obfuscation of design intent in object-oriented applications," in *DRM '03*. ACM, Oct. 2003, pp. 142–153.

[17] A. Lakhotia, E. U. Kumar, and M. Venable, "A method for detecting obfuscated calls in malicious binaries," *IEEE Transactions on Software Engineering*, vol. 13, no. 11, pp. 955–968, 2005.

[18] W. Zhu, C. Thomborson, and F.-Y. Wang, "A survey of software watermarking," in *ISI 2005*, ser. LNCS, vol. 3495, May 2005, pp. 454–458.

[19] ——, "Application of homomorphic function to software obfuscation," in *WISI 2006*, ser. LNCS, April 2006.