Applications of Homomorphic Functions to Software Obfuscation.

CONFERENCE PAPER · JANUARY 2006

DOI: 10.1007/11734628_18 · Source: DBLP

3 AUTHORS:

William Zhu
MinNan Normal University
149 PUBLICATIONS **2,522** CITATIONS
SEE PROFILE

Clark David Thomborson
University of Auckland
127 PUBLICATIONS **5,285** CITATIONS
SEE PROFILE

jj Wang
The Hong Kong University of Science and T…
201 PUBLICATIONS **3,600** CITATIONS
SEE PROFILE

Applications of Homomorphic Functions to Software Obfuscation*

William Zhu[1], Clark Thomborson[1], and Fei-Yue Wang[2,3]

[1] Computer Science Department, The University of Auckland, Auckland, New Zealand
fzhu009@ec.auckland.ac.nz, cthombor@cs.auckland.ac.nz
[2] The Key Laboratory of Complex Systems and Intelligent Science,
Institute of Automation, The Chinese Academy of Sciences, Beijing 100080, China
[3] Systems and Industrial Engineering Department,
The University of Arizona, Tucson, AZ 85721, USA
feiyue@sie.arizona.edu

1 Introduction

As various computers are connected into a world wide network, software is a target of copyright pirates, attackers, or even terrorists, as a result, software protections become 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 [1], etc. Software obfuscation [2] is one of these measures to protect software from unauthorized modification by making software more obscure so that it is hard for potential attackers to understand the obfuscated software. There are several algorithms of software obfuscation such as layout transformation, computation transformation, ordering transformation, and data transformation [2]. Variable transformation is a major method of data transformation to transform software into a new semantically equivalent one that is hard for attackers to understand the true meaning of variables in software.

Chow et al. applied residue number technique, an approach used in hardware design, high precision integer arithmetic, and cryptography, to software obfuscation by encoding variables in the original program to hide the true meaning of these variables [3], but part of the technique proposed there is incorrect. In order to compensate this drawback, in paper [4], we proposed homomorphic functions, developed an algorithm for division by several constants based on homomorphic functions, and applied them to variable transformation.

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 attacker to modify them. In this paper, we apply homomorphic functions to obfuscate arrays in software through array index change, array index and dimension change, array folding, and array flattening. As said in [2], by adding the data complexity in the program, these methods can make a program much more difficult to understand and reverse engineer. We are investigating the security of these applications.

* Research supported in part by the New Economy Research Fund of New Zealand.

H. Chen et al. (Eds.): WISI 2006, LNCS 3917, pp. 152 – 153, 2006.

2 Application of Homomorphic Functions to Array's Change

We describe four methods to apply homomorphic functions to software obfuscation: index change, index and dimension change, array folding and array flattening.

1. Index change
For an array A[n], firstly, find an m such that m > n, and n and m are relatively prime, then change A[n] into B[n] and the element A[i] is turned into b[i*m mod n].
2. Index and dimension change
For an array A[n], firstly, find an m such that m > n, and n and m are relatively prime, then change A[n] into array B[m] and the element A[i] is turned into b[i*n mod m].
3. 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 <= i < n and C[i] undefined for others.

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 <= i < m.

Replace any A[i] with B[i mod m_1, i mod m_2] in the unobfuscated program.
4. Array flattening
For a 2-dimensional 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 <= m_2 and n_2 <= 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 <= i < m_1 and 0 <= 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 <= i < m.

Replace any A[i, j] with B[(i*k_1 + j*k_2) mod m] for 0 <= i < n_1 and 0 <= j < n_2.

3 Conclusion

We propose applications of homomorphic functions to obfuscate arrays in software.

References

1. W. Zhu, C. Thomborson, F.-Y. Wang, A survey of software watermarking, LNCS 3495, 2005, pp. 454-458.
2. C. Collberg, C. Thomborson, D. Low, A taxonomy of obfuscating transformations, Tech. Report, No.148, Dept. of Computer Sciences, Univ. of Auckland, 1997.
3. Chow, et al, Tamper resistant software encoding, US patent 6594761 (2003) 1-32.
4. W. Zhu, C. Thomborson, A provable scheme for homomorphic obfuscation in software security, in: The IASTED International Conference on Communication, Network and Information Security, CNIS'05, 2005, pp. 208-212.