Protecting Programs from Hostile Environments: Encrypted Computation, Obfuscation, and Other Techniques by Luis Sarmenta

Dengfeng Gao

October 4, 1999

1 Introduction

This paper presents and discusses techniques for protecting executable code from malicious users trying to reverse engineer or modify the code for their own purposes. Three main techniques are discussed:

- Encrypted computation.
- Obfuscation.
- Time-limited blackbox security.

Based on the three techniques, the author concludes with some suggestions on possible directions for future research.

2 Overview

Ideally, the programs must be protected in such a way that a host executing a program it didn't create:

- Cannot read, or at least cannot understand, the program's code and data, and
- Cannot modify the program's code and data.

This is called blackbox security. Such a security scheme would have many applications:

- Protection of intellectual property
• Protection of mobile agents
• Market-based parallel computing

The three different techniques will be discussed are possible solutions to these problems.

3 Encrypted Computation

Encrypted computation seeks to provide true blackbox security by employing provably secure cryptographic techniques to allow an untrusted host to perform useful computation without understanding what it is computing. Sander and Tschudin have developed techniques for the encrypted computation of polynomials, which includes:

• Polynomial evaluation using homomorphic encryption schemes - they point out an additively homomorphic encryption scheme can be used to make encrypted computation of polynomials.

• Composition-based approach - using function composition to prevent the host discovering the secret key, since the decomposition of multivariate rational functions is a hard problem in general.

• Using randomization and redundancy - to detect the changes made by the host.

Their work gives us hope of achieving true blackbox security with theoretically provable strength. However, their work does not yet seem to be useful in real applications for several reasons:

• Limited functionality - polynomials and rational functions
• Poor efficiency - encryption and decryption overhead
• Impracticality for function hiding and market-based computation

4 Obfuscation

Obfuscation uses heuristic techniques to modify the program by transforming it into a "messy" form that is very hard to understand, but performs the same function as the original program.
Collberg, Thomborson and Low have classified obfuscation transformations into four general types: layout, control, data, and preventative transformations. They further identify four main criteria that can be used to measure the quality of an obfuscation transformation: potency, resiliency, stealth, and cost. A key tool in implementing these transformations is the idea of opaque variables and opaque predicates which have some property that is known a priori to the obfuscator, but difficult for a deobfuscator to deduce.

The significant advantage in their approach over those of others is, they identify some concrete formal metrics of program complexity from software engineering research and propose to measure them and use them in their obfuscation program. However, The effectivity of their proposed techniques remains to be seen. The main weakness of obfuscation is the difficulty of assuresing how much these transformations really make it harder for a human user (fully-armed with deobfuscation tools) to understand the obfuscated code. Often it is not necessary to discover all the rules and break all the opaque predicates to successfully attack a piece of code. Another problem is obfuscator alone does not protect against sabotage.

5 Time-limited Blackbox Security

Time-limited blackbox security recognizes the shortcoming of obfuscation by restricting the definition of blackbox security as follows: programs must be protected in such a way that a host executing a program it didn't create:

- Cannot read, or at least cannot understand, the program's code and data, and
- Cannot modify the program's code and data
- - for a certain known time interval.
- After this time interval, attacks are possible, but these attacks would have no effects.

Hohl presents this approach as a way to protect mobile agents. This kind of security can be achieved by first "messing-up" the agent code and data to make it hard to analyze, and then attaching an expiration data to indicate its protection interval. One mess-up technique proposed by Hohl but not by Collberg et al. is that of deposited keys.

There are some open problems:
• A need for a standard global time.

• How to decide the protection interval.

Hohl's list of mess-up algorithms can benefit a lot from incorporating Collberg et al.'s obfuscation transformations. The idea of deposited keys can be an advantage and a disadvantage. The disadvantage is that it requires communication with a trusted server, and can thus increase execution time and limit scalability. The advantage is that one can rely on more information-theoretic, and not just complexity-based security against attacks.

6 Suggestions for Future Research

Some ideas for further research includes the following:

• Holding a cracking contest to measure effectivity - This would be similar to the successful decryption contests held by RSA.

• Redundancy and Randomization - Further research in this area may also benefit from the results of research in digital watermarking.

• Combining encrypted computation and obfuscation - Some possible approaches include: obfuscating encrypted functions, encryption as part of the obfuscation process, and obfuscation as encryption.

Since obfuscation can be broken, given some information about the original program, obfuscation as encryption might need to mask the statistical pattern of the original program. One way to mask the statistical pattern is to interleave different and independent computation. If two such computations have different statistical pattern, the host will be confused. Whether these and other techniques would be enough to make obfuscation feasible as a way of doing encrypted computation is still an open question, but it looks very promising and thus deserves further investigation.

7 Conclusions

In the end, a successful solution to this problem will probably be achieved by combining all these techniques - that is, by using parameterized obfuscation as an encryption scheme (with randomized parameters as the private keys) to prevent the malicious host from understanding and manipulating the code, and by using checksums and watermarks to detect attempts to change the code.