

THE ENCODER SOLUTION TO IMPLEMENTING TAMPER RESISTANT SOFTWARE

J.R. NICKERSON, S.T. CHOW, H.J. JOHNSON, AND Y. GU

Cloakware Corporation
Kanata Square
260 Hearst Way, Suite 311
Kanata, ON K2L 3H1

ABSTRACT. Tamper Resistant Software (TRS) offers a temporal window of protection to software executing on a hostile host. TRS in various implementations has been proposed over the past five years. Despite being a uniquely powerful solution to the survivability of information in this environment, TRS has not seen widespread use on any front. We believe that the lack of a method to transparently and automatically convert source code from a native form into a TRS form has been the major impediment to common usage of TRS and the protection it affords. In this paper we illustrate an encoder solution to this problem, describe its operation and use.

1. INTRODUCTION

In the complementary sense to the classic example of protecting a user from hostile code, is the need to protect software executing on a hostile host. Some examples of applications operating in hostile environments are digital rights management applications and mobile agents. The common themes shared by these two applications are the need to protect the executable from being modified and/or to prevent some secret embedded in software from being extracted by an attacker. There are now web sites (see [5]) devoted to cracking in this sense and branch jamming for improving the scores of gamers.

Our goal is to make it difficult for attackers to reverse engineer the hidden data from the code or change the outcome (branch jamming for instance). Although TRS is complementary to cryptography, it will in many cases be the last line of defense against reverse engineering attacks. TRS should be distinguished from Obfuscated Software (ObS) which only seeks to resist disassembly. TRS on the other hand must appear to be robust to changes but in reality be fragile to any changes. If an attacker changes the code it should fail but not in a manner that is obvious or conveys information that is useful to the attacker. Thus the generation of TRS should obviate reverse engineering especially through automated means.

TRS or the subset ObS has been proposed in several variations:

1. Auscsmith [1] which is one of the earliest works on TRS.
2. Collberg et al [3] and [4] used opaque predicates which like the Auscsmith work [1] resisted static analysis.
3. Mambo [6] had an earlier work as well.
4. Wang et al [9] used pointer aliasing to generate TRS, which like the aforementioned works, resisted static analysis.
5. A later work by the latter three authors of this paper, published in the PCT [8], seeks to generate TRS with combined schemes (described in [7] and partially analyzed in [2]) that are resistant to static and dynamic analysis.

There are however impediments to implementing TRS software:

1. The lack of good security metrics.
2. Inability to Generate multiple instances of the original that are functionally equivalent to the original code.
3. The lack of an automatic TRS generator.

In fact the authors of “*A Taxonomy of Obfuscating Transformations*” argue that “*automatic code obfuscation is currently the most viable method for preventing reverse engineering*”, [3].

Item 1 above will be addressed elsewhere. Item 2 is a desirable feature in that if an attack is successful on one instance, but cannot be automated, then it will take an equivalent amount of work to break another instance. Based on the authors’ newer data encodings [7], the number of instances that can potentially be created is cryptographically large. Solving the third problem above will of course solve the multiple instance impediment. The purpose of this paper is to address item 3, above which is detailed in the following section. Performance issues of the encoder and of the transformed code are discussed in Section 3. Conclusions and Future Work will conclude this paper.

2. AUTOMATED GENERATION OF TRS

In Figure 1, a block diagram of the encoder currently in use, is given. This encoder (approximately 300,000 lines long, written in Java for portability), takes ANSI C program code and produces a “cloaked” or TRS derivative of the program that is functionally equivalent. The process starts with:

1. The Prefab Compiler, which compiles the language specification and generates Java class files from Morphers or data transformations (see [7]).
2. The original C source is optionally sent through a C to C optimizer and:
3. Through a translator (C2F) into a language called “fabric”. The reason for this layer is that fabric is a complete language description and other language to fabric translators may be used. That is, we can have C++ to fabric or Java to fabric converters, and have the same encoder

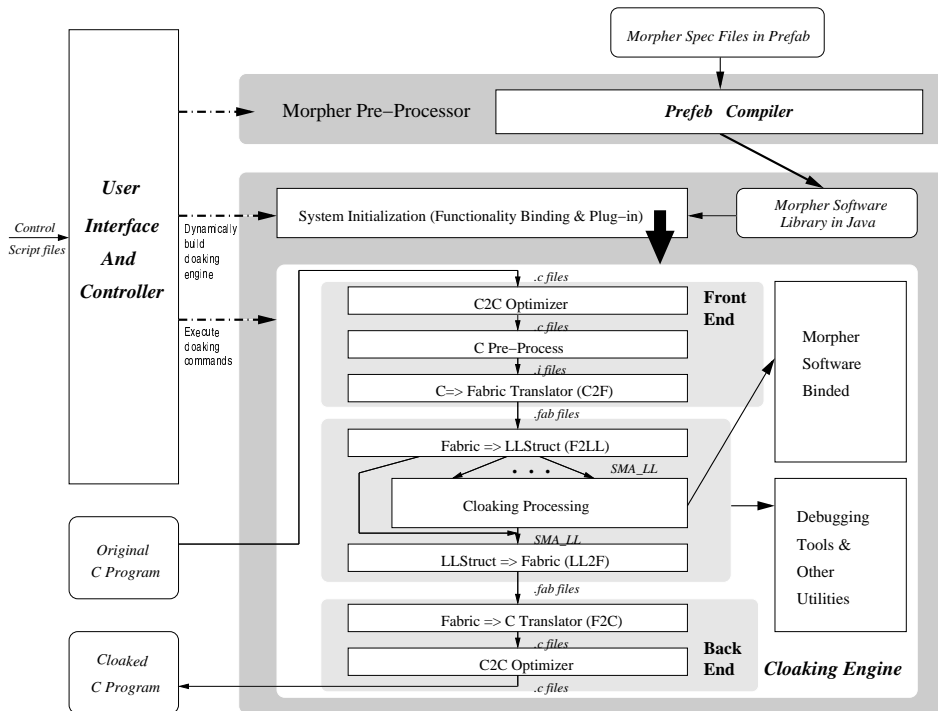


FIGURE 1. Cloakware Encoder

accommodate different languages. After the fabric language conversion the encoder converts the output from this step into:

4. LLStruct or Low Level Structure language (F2LL). The LLStruct is required to put in new operators, new variable interpolators, and create op codes for the morphers. It is at this point the actual “Cloaking Process” or TRS generation takes place. The language conversion process now takes place in reverse:
5. LLStruct to fabric (LL2F) and:
6. Fabric to C (F2C) conversion.
7. A pass through an optimizer and the resultant cloaked or TRS representation of the original C program is generated which is homomorphically equivalent to the original.

It should be noted that the encoder as described, can selectively convert segments of a code into the equivalent TRS form while leaving other segments unencoded. This minimizes code expansion and any speed penalties. Further this allows the encoder operator to place security where it is needed as opposed to other areas like a GUI where security may be less of a concern.

3. PERFORMANCE AND ENCODER USAGE

In-house work with the encoder on clients' production C code shows that cloaking 200 line segments with the encoder takes from thirty (30) seconds to one (1) minute. This timing of course, depends on the data transformations and in the worst case is expected to take no more than five (5) minutes. The resultant TRS cloaking is typically 2-3 times the size of the original and runs approximately 4-5 times slower.

4. CONCLUSIONS AND FUTURE WORK

The encoder as described represents approximately twenty (20) man years of development work. It works on transforming ANSI C codes and is constantly being upgraded to include new data encodings. Later this year modifications to include control flow encoding [7], [2] will be made. These new encodings will provide a greater measure of temporal security. With the addition of a debugger, the encoder will allow us to quickly and easily transform source code into TRS and make this new security paradigm ubiquitous.

REFERENCES

- [1] D. Aucsmith, *Tamper Resistant Software: An Implementation*, Information Hiding: 1st Int. Workshop (Lecture Notes in Computer Science), vol. 1174, R.J. Anderson, Editor, Springer Verlag, 1996, Berlin, pp.317-333.
- [2] S. T. Chow, Y. Gu, H.J. Johnson and V.A. Zakharov, *An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs*, Paper accepted at the International Security Conference 2001, Malaga, Spain.
- [3] C. Collberg, C. Thomborson and D. Low, *A Taxonomy of Obfuscating Transforms*, Technical Report Number 148, Dept. of Computer Science, University of Auckland, New Zealand, July 1997. Postscript version of this paper is available at: <http://www.cs.auckland.ac.nz/~collberg/Research/Publications/CollbergThomborsonLow97a/index.html>
- [4] C. Collberg, C. Thomborson and D. Low, *Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs*, Dept. of Computer Science, University of Auckland, New Zealand, 1998. PDF version of this paper is available at: <http://www.cs.arizona.edu/~collberg/Research/Publications/CollbergThomborsonLow98a/LETTER.pdf>
- [5] Fravia, *Fravia's Pages of Reverse Engineering*, see <http://www.woodmann.com/fravia/>
- [6] M. Mambo, T. Murayama and E. Okamoto, *A tentative approach to constructing tamper-resistant software*, 1997 New Security Paradigms Workshop, ACM Publication 0-89791-986-6/97/9.
- [7] J.R. Nickerson, S.T. Chow and H.J. Johnson, *Tamper Resistant Software: Extending Trust into a Hostile Environment*, Multimedia and Security Workshop at ACM Multimedia 2001, Ottawa, October 5, 2001.
- [8] <http://www.delphion.com/>, PCT Patent Applications: WO077597A1, WO077596A1, WO114953A1.
- [9] C. Wang, J. Hill, J. Knight and J. Davidson, *Software Tamper Resistance: Obstructing Static Analysis of Programs* University of Virginia Technical Report available in PDF form from <http://citeseer.nj.nec.com/wang00software.html>