On Source Code Transformations for Steganographic Applications

Geoffrey C. Hulette
Computer and Information Science Department
University of Oregon
Eugene, OR, USA
ghulette@cs.uoregon.edu

John Solis
Scalable and Secure Systems Research Department
Sandia National Labs
Livermore, CA, USA
jhsolis@sandia.gov

***Abstract*—The amount of publicly available source code on the Internet makes it attractive as a potential message carrier for steganographic applications. Unfortunately, it is often overlooked since embedding information in an undetectable way is challenging. We investigate term rewriting as a method for embedding messages into programs via transformations on source code.**

We elaborate on several possible transformation strategies and discuss how they might be applied in a steganographic setting. We continue with a discussion on (a) the implications and trade-offs of preserving semantic properties, (b) the relationship between messages and transformations, and (c) how to incorporate existing natural language processing techniques. The goal of this work is to elicit constructive feedback and present ideas that stimulate future work.

***Keywords*-steganography; source code transformations; term rewriting;**

I. Introduction/Motivation

The field of steganography studies techniques for embedding information into inconspicuous carriers such that outside observers cannot easily detect the presence of this information. One typical application scenario is the discreet transfer of secret messages between two parties. For example, in a country forbidding anti-government rhetoric, its citizens could communicate by transmitting sensitive messages embedded in image files. A second scenario, document watermarking, is a digital rights management method for determining if copies of certain media files are unauthorized. In both scenarios, the same steganographic techniques can, in principle, be applied to embed watermarks or secret messages into a variety of types of documents. For example, information may be embedded into bitmapped images by manipulating the low-order bits of each pixel.

Source code documents are often overlooked as a potential steganographic carrier. This oversight is partly due to the challenge of preserving semantic correctness while simultaneously making the embedded information difficult to detect. For instance, adding dummy methods never referenced by the original source preserves program semantics, but will likely be identified as suspicious by an analyst. Why, therefore, is source code an attractive carrier?

The number of publicly available source code repositories has exploded with the advent of the Internet and open source software. The total amount of available source code is staggering once we consider everything from large repository websites to individual repositories and instructional websites with sample source code. PlanetSourceCode.com alone claims a database containing over 29 million lines of code [7]. This prevalence makes source code an attractive information carrier because it is likely to be overlooked or dismissed by analysts.

In this paper, we investigate novel techniques for embedding messages into program source code using transformations based on term rewriting. We present three different approaches, and show how each approach might be applied in a steganographic setting. We continue with a discussion on preserving semantic properties, the relationship between messages and transformations, and how to incorporate existing natural language processing techniques. Finally, we conclude with topics for potential future work.

II. Term Rewriting

Our method encodes a secret message in a log of transformations applied to a given source code text. Crucially, transformations must be deterministic so that the message can be reliably recovered. At the same time, transformations must be flexible enough to encode a variety of messages. *Term rewriting* [8] provides a framework and theory for program transformations that is flexible enough for our purposes, and can be restricted to ensure determinism.

For our application, *terms* are a structured representation of source code in a given programming language, e.g., an abstract syntax tree. There are typically many different ways to encode a given language as terms – deciding this representation is usually the first step in term rewriting applications.

Rewriting is a procedure for transforming one term into another by applying a set of rules. Conceptually, rules give an identity relationship between terms. Terms are gradually transformed, step-by-step, by applying rules non-deterministically at any sub-term where they are valid.

Given a set of rules, we must then provide a *strategy* to determine the order and context in which to apply them. Different systems approach this problem in different ways. One approach, convenient for steganography, is explicitly

directing rule application through rule *expressions*. System S [9] is a formal semantics for such expressions, and is briefly described below.

A. System S

The basic unit of an expression in System S is a *primitive rule*. Primitive rules contains two components: the first is a pattern that is matched against the top-level (outermost) term, and the second is another pattern that replaces the input term. Rule application may "fail" if the first pattern does not "match" the input. If the input does match, then the rule application succeeds, and the output is the second pattern of the primitive rule. Primitive rules will typically allow variables in the pattern, which are bound as a side-effect of a successful match and can be used in the rule's result term.

System S allows the construction of rule expressions using a set of combinators. For example, the *sequence* combinator allows us to apply two rule expressions in sequence, with the second acting on the output of the first. Similarly, the *left-biased choice* combinator will attempt to apply the first rule expression, and resort to the second just in case the first fails. Other combinators allow recursion, traversal of subterms, and so on. Crucially, System S permits us to preclude non-deterministic strategies by eschewing the pure *choice* combinator, as described in [9].

To support steganographic applications, we extend System S with tracing semantics to record rule applications. First, we assert that primitive rules must be labeled. Second, successful application of a primitive rule is logged (i.e. its label recorded) sequentially, in the order of application. The labels of the rules thus form the alphabet for the obfuscated message, and the trace itself forms the message string. We omit the formal semantics due to lack of space.

System S treats failure as a special case. A failed program indicates that no rules were successfully applied and that the program was not transformed. Failures are represented by a single distinguished token, in lieu of a transformed program, and there is no trace output. For the purposes of steganography, failure could be considered a valid message, albeit with exactly one form.

III. Methods

Now we will examine how the extended System S machinery might be put to steganographic use. We consider three scenarios. In each of following, let f, g, etc. represent expressions in System S enhanced with tracing, let x, y, etc. range over term encodings of programs in a given language, and let m, n etc. range over trace outputs. We will write $x \xrightarrow{f} (x', m)$ to say that the application of rewriting program f to the term representation of source code x is successfully rewritten to another term x' with trace m. We write $x \xrightarrow{f} \downarrow$ to indicate that the transformation failed.

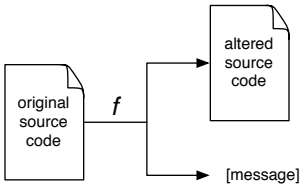

Figure 1. One-way transform

A. One-way transform

In the first scenario, we construct a transformation f and a source code term x such that $x \xrightarrow{f} (x', m)$ where x' is a valid transformed program and m is the desired obfuscated message. We expect that f would be communicated between sender and recipient through some back channel, and that the source code x would be available publicly or transmitted through some low security channel. To recover the message the recipient simply evaluates the transformation f on x. See Figure 1 for a graphical illustration.

This approach is conceptually simple, but has the drawback that f must be communicated separately.

B. Two-way transform

In the second scenario, we construct a transformation f and term x as before, such that $x \xrightarrow{f} (x', m)$ where x' is a valid transformed program and m is the desired message. We add the constraint that there must exist a transformation f^{-1} such that $x' \xrightarrow{f^{-1}} (x, m^{-1})$, where m^{-1} is the mirror (reversed) string m. Conceptually, f^{-1} is the inverse transformation of f. Under certain conditions and restrictions (beyond the scope of this paper), given f we can produce f^{-1} or vice versa.

The advantage of this approach over the first is that the original source code need not be distributed. Instead, the transformed code x' can be distributed, and the message recovered by transforming it with f^{-1} and then reversing the resulting trace to recover the message (see Figure 2). This may be desirable in cases where the original source code represents sensitive information. In this case, our method may be used to obfuscate the original source code in addition to the message itself.

C. Recover transformation from differences

In our third and final scenario, the transformation f is not communicated, but instead is recovered by examining the differences between two terms x and x' constructed so that $x \xrightarrow{f} (x', m)$ (see Figure 3). Notice that in this case, f must be unique – that is, if $x \xrightarrow{f} (x', m)$ and $x \xrightarrow{g} (x', n)$ then we require that $f = g$ (and, consequently and crucially, that $m = n$).

The advantage of this approach is the removal of a second channel to communicate f, since it can be recovered from

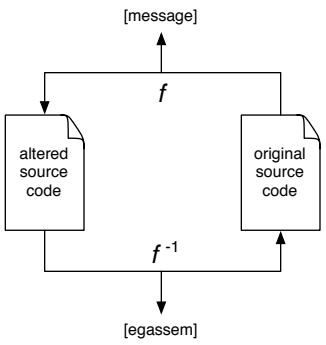

Figure 2. Two-way transform

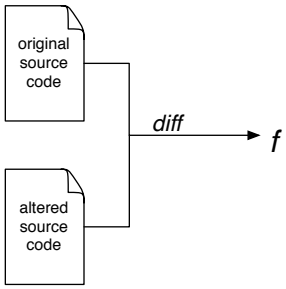

Figure 3. Recover transform function

source code alone and then used to transform x and recover m. Of course, in this case, both the original and transformed versions of the source code are required to recover f. This approach could be useful in contexts where multiple versions of the same code document might be expected, e.g., in a version-controlled source repository.

While this scenario is interesting in theory, it may be difficult in practice to recover a non-trivial f from x and x', as well as to ensure uniqueness of f. This is a topic of ongoing research.

IV. Discussion

A. Semantic Properties

It may be useful to preserve the semantics of programs under transformation. This would be desirable, for example, if we foresaw the source code being inspected for legitimacy – in this case, preserving or mostly preserving the semantics of the code could make it appear that the transformation was applied in the service of software development rather than steganography. Conversely, a program that has been transformed in such a way as to render it broken, inefficient, or nonsensical may invite unwanted scrutiny.

Ensuring preservation of program semantics under transformation is quite difficult, even without ulterior steganographic motives. We expect that this will be an interesting area for future work.

B. Constructing Messages

We have not fully addressed the question of how to construct a non-trivial transformation f that induces a particular message m when given a fixed source code x. Ideally, since f must be communicated separately, it should be flexible enough to be reusable on many different source code documents to produce different messages as needed. That is, once f is constructed, it should ideally remain fixed or at least change infrequently. In general, we know that it is trivial to construct a pair f and x given some desired m, but constructing an f that is flexible in this sense is non-trivial.

C. Combining with Natural Language Processing

The methods discussed above naturally lend themselves to being combined with natural language processing (NLP) techniques. The Semantilog Project [10] is a comprehensive bibliography of linguistic steganographic techniques, both theoretical and applied, for a number of languages, including English, Japanese, Chinese, Persian, and Arabic. In our context, we treat NLP steganography as a black box capable of embedding information into the comments and other documentation found in source code files.

The basic idea is to use existing NLP steganography tools to create a second information channel in the source text to augment the term rewriting channel. The two independent channels can be used to (a) duplicate the embedded message or (b) split the message using cryptographic secret sharing.

The first approach attempts to improve message recoverability. Ideally, the techniques applied to each individual channel should be robust, i.e., messages are recoverable despite small changes to the source. However, a second information channel provides another opportunity for recovery if the first channel is lost.

In the second situation, cryptographic secret sharing should make the source code more robust to statistical analysis looking for variations in entropy. Each channel, considered independently, does not reveal any information since the secret share is itself indistinguishable from random. Messages can only be recovered when both shares are recovered and successfully combined.

An alternate strategy is to make the source code documentation completely separate from the source code itself. In this situation, an analyst must correctly identify and associate the documentation file with its corresponding source code. Note that this mapping can (and should be) independent and random, e.g., the documentation for source file X is mapped to source code Y. This mapping can be identified using a pseudo-random permutation keyed by a shared secret key, established a-priori, between the communicating parties. These shared secrets do not defeat the goals of this paper. Steganography attempts to hide the fact that information is embedded in some carrier and recoverable by anyone who knows the recovery method. This is in contrast to

cryptography where information is obviously encrypted but impossible to recover.

D. Fundamental Limitations

There are fundamental limits to the amount of protection that any steganographic application can provide. As shown by the results in [**?**], it is impossible to construct an obfuscator (and by extension a software watermarker) that makes some information about a program "unintelligible". A determined adversary will locate information embedded by a steganographic application if provided enough samples. In our scenario, transformed programs could be compiled with a highly optimizing compiler and then immediately decompiled. The difference between the decompiled source code and original program may reveal that additional information has been embedded. Our goal, as with any steganographic scheme, is to make transformed programs appear as inconspicuous as possible in hopes of escaping further scrutiny.

V. Related Work

The field of steganography has been well studied with many techniques developed for embedding information into specific carriers, including video [1], audio [3], and natural language text [10]. The sub-field of *software watermarking* investigates techniques that discourage illegal duplication by allowing authorities to prove ownership of some particular piece of software. This can be accomplished through register allocation patterns [13], dynamic path execution [14], and spread-spectrum techniques for robust watermarks [16]. These approaches do not consider the originating source code as a possible carrier.

In contrast to watermarking, we are not interested in detecting illegal duplication. Instead, we would like to investigate how source code, and more specifically source code transformations, can be used to embed secret messages or information.

VI. Future Work and Conclusion

As a next step, we plan to implement a system that incorporates the basic capability described in Section III-A. Such an implementation would allow us to answer some practical open questions, such as, what is the total amount of information each transformation is capable of embedding? A concrete implementation would also help determine the feasibility of incorporating existing NLP steganographic approaches as a second information channel. This may turn out to be impossible if the information capacity of one channel greatly exceeds that of the other.

The most challenging step will be to develop the theory to support invertible term rewriting transformations and to recover unique functions from differences between source codes. This theory will be a prerequisite for the application scenarios described in Sections III-B and III-C. It may be difficult or impossible to guarantee that a transformation is invertible, or that the inversion is unique. However, the three approaches we have identified merit further investigation. We hope that this paper elicits constructive feedback and stimulates future work in this area.

Acknowledgments

We would like to thank the anonymous reviewers of the Web Intelligence for Information Security Workshop 2011 (WIIS'11) for insightful comments and suggestions.

This work was funded by the Laboratory Directed Research and Development (LDRD) program at Sandia National Laboratories. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] H. Noda, T. Furuta, M. Niimi, and E. Kawaguchi, "Application of BPCS steganography to wavelet compressed video," in *ICIP '04*, vol. 4, Oct 2004, pp. 2147–2150.

[2] K. Gopalan, "Audio steganography using bit modification," in *ICME '03*, vol. 1, July 2003, pp. 629–632.

[3] N. Cvejic and T. Seppanen, "Increasing robustness of LSB audio steganography using a novel embedding method," in *ITCC '04*, vol. 2, April 2004, pp. 533–537.

[4] M. J. Atallah, V. Raskin, M. Crogan, C. Hempelmann, F. Kerschbaum, D. Mohamed, and S. Naik, "Natural language watermarking: Design, analysis, and a proof-of-concept implementation," in *IHW '01*. Springer-Verlag, 2001, pp. 185–199.

[5] R. El-Khalil and A. D. Keromytis, "Hydan: Hiding information in program binaries," in *Information and Communications Security*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 3269, pp. 287–291.

[6] "SourceForge.net: Find, create, and publish Open Source software for free," http://www.sourceforge.net/.

[7] "PlanetSourceCode.com: The largest public source code database on the internet," http://www.planet-source-code.com/.

[8] F. Baader and T. Nipkow, *Term Rewriting and All That*. Cambridge University Press, 1998.

[9] E. Visser and Z. el Abidine Benaissa, "A core language for rewriting," *Electronic Notes in Theoretical Computer Science*, vol. 15, pp. 422–441, 1998.

[10] R. Bergmair, "A comprehensive bibliography of linguistic steganography," The Semantilog project: http://www.semantilog.org/biblingsteg/.

[11] R. Anderson and F. Petitcolas, "On the limits of steganography," *Selected Areas in Communications, IEEE Journal on*, vol. 16, no. 4, pp. 474–481, 1998.

[12] C. Cachin, "An information-theoretic model for steganography," in *Information Hiding*, ser. Lecture Notes in Computer Science. Springer, 1998, vol. 1525, pp. 306–318.

[13] G. Myles and C. Collberg, "Software watermarking through register allocation: Implementation, analysis, and attacks," in *ICISC 2003*, ser. Lecture Notes in Computer Science. Springer, 2004, vol. 2971, pp. 274–293.

[14] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Stepp, "Dynamic path-based software watermarking," *SIGPLAN Not.*, vol. 39, pp. 107–118, 2004.

[15] R. Venkatesan, V. V. Vazirani, and S. Sinha, "A graph theoretic approach to software watermarking," in *IHW '01*. Springer-Verlag, 2001, pp. 157–168.

[16] J. P. Stern, G. Hachez, F. Koeune, and J.-J. Quisquater, "Robust object watermarking: Application to code," in *IH '99*. Springer-Verlag, 2000, pp. 368–378.