SOURCE CODE WATERMARKING BASED ON FUNCTION DEPENDENCY ORIENTED SEQUENCING

Gaurav Gupta and Josef Pieprzyk
Centre for Advanced Computing - Algorithms and Cryptography,
Department of Computing, Macquarie University
NSW - 2109, Australia
{ggupta,josef}@ics.mq.edu.au

Abstract

In the current market, extensive software development is taking place and the software industry is thriving. Major software giants have stated source code theft as a major threat to revenues. By inserting an identity-establishing watermark in the source code, a company can prove it's ownership over the source code. In this paper, we propose a watermarking scheme for C/C++ source codes by exploiting the language restrictions. If a function calls another function, the latter needs to be defined in the code before the former, unless one uses function pre-declarations. We embed the watermark in the code by imposing an ordering on the mutually independent functions by introducing bogus dependency. Removal of dependency by the attacker to erase the watermark requires extensive manual intervention thereby making the attack infeasible. The scheme is also secure against subtractive and additive attacks. Using our watermarking scheme, an n-bit watermark can be embedded in a program having n independent functions. The scheme is implemented on several sample codes and performance changes are analyzed.

1. INTRODUCTION

Watermarking and fingerprinting are copyright protection techniques. While watermarking deals with identify the actual owner, fingerprinting deals with the more complex task of identifying traitor(s). The notable advancement in watermarking and fingerprinting during the past decade is attributed to the threat posed by electronic distribution of multimedia objects including software over the Internet. Transfer of digital data on the internet has led to piracy, tampering and duplication of genuine software. Software watermarking addresses two security concerns,

1. **Copyright protection**: Protecting the right of the author on the software.
2. **Software authentication, Tamper detection**: Protecting the validity and integrity of the software.

Different kinds of watermarking techniques are used to satisfy these requirements. *Robust* watermarking inserts a watermark which is extremely hard to eliminate, even with a strong attack. *Fragile* watermarking inserts a watermark which is very easily destroyed by the slightest attacks. Hence robust watermarking is used for copyright protection while fragile watermark is utilized for software authentication and tamper detection. Based on embedding process, the software watermarking schemes can be classified as follows:

1. **Graph-based/branch-based software watermarking**: Software G_s and watermark code G_w are converted to graphs with sequential code as nodes and function calls as edges between the nodes. G_s and G_w are connected by inserting additional edges (jump instructions or function calls or branch statements) resulting into $G = G_s + G_w$. Goal of the attacker is to find the cut C that separates the two graphs and recover G_s.

   Venkatesan et al. [12] proposed the first graph-based software watermarking scheme. The software and the watermark code are converted into digraphs and new edges are introduced between the two by adding function calls between the software code and watermark code. Error-correcting capabilities were missing from the scheme and it was also susceptible to attacks that reorder the instructions and add new function calls. Another problem was the random walk (meaning that the next node to be added in the watermarked software graph should be selected randomly) mentioned in the paper is not completely "random". Out of the remaining nodes, if N_s belong to software graph and

978-0-7695-3278-3/08 $25.00 © 2008 IEEE
DOI 10.1109/IIH-MSP.2008.188

N_w belong to watermark graph, the probability of next node being from the watermark nodes is $\frac{N_w}{N_w+N_s}$ and being from the software nodes is $\frac{N_s}{N_w+N_s}$. This gives excessive information to the attacker. Alternatively, a pseudo-random permutation of the nodes to be visited can be generated. For further details in graph-based software watermarking, please refer to [1, 2, 3, 11]. Gupta and Pieprzyk broke the branch-based software watermarking scheme proposed in [7] and later presented an improved watermarking model in [5] secure against the earlier proposed debugging attacks in [4].

2. **Register-based software watermarking**: Watermark bits are encoded in the registers used for storing variables. Certain higher level language blocks by inline assembly code that controls which registers store which variables. Goal of the attack is to re-allocate variables in different registers to distort the encoding. Register-based software watermarking based on the QP algorithm (named after authors Qu and Potkonjak) [9, 10] has been proposed in [6]. It changes the registers used to store variables depending on the variables required at the same time. The scheme is unable to survive register reallocation and recompilation attacks as well as secondary watermarking attacks. Also, inserting bogus methods renders the watermark useless by changing the interference graph.

3. **Thread-based software watermarking**: Nagra et al. [8] propose embedding watermark in the sequence of threads that execute a particular piece of code. If there are 2 threads; T_1, T_2, then T_1 executing code c_1, c_2 encodes $(00)_2$, T_2 executing code c_1, c_2 encodes $(01)_2$, T_1 executing c_1 and T_2 executing c_2 can encode watermark $(10)_2$ and T_1 executing c_2 and T_2 executing c_1 can encode watermark $(11)_2$. Though, changing threads that execute a particular piece of a code, merging or integrating threads and codes would destroy the watermark.

4. **Obfuscation-based software watermarking**: Only object-oriented softwares can be watermarked using this class of watermarking. Class C having functions $\{f_1, f_2, \ldots, f_n\}$ is split into k subclasses $\{C_1, C_2, \ldots, C_k\}$ and the watermark is encoded in the functions carried out by these subclasses.

We present a model of watermarking specifically for C/C++ source codes. Watermarking source codes is required to successfully lodge a claim on a software and sue any competitor who steals the source code. The Company A who claims that it has developed software S having source code C and company B who, according to A, has stolen code C to develop S' produce their source codes to Judge J along with secret key K provided by A and the watermark extracted determines the ownership. Since both the parties need to submit their source codes, the watermarking scheme has to be non-blind. Certain restrictions are placed by these programming languages on the sequencing of functions in the program. In terms of programming language design and compiler requirements, it is required that a function f_b be defined before function f_a, if f_a calls f_b. The logical interpretation is that the compiler needs to understand the *meaning* of f_b before it can be called.

2. PROPOSED SCHEME

The central idea of our watermarking scheme is to modify the sequence in which functions in F_i are defined in the program such that the watermark is encoded in the *sequencing*.

Notations used:

- Let the set of functions in the program be F that equals $F_d \cup F_i$ where,
  - F_i is the set of independent functions, such that any function f_i in F_i does not call any other function.
  - F_d is the set of dependent functions that call other function(s).
- For any two functions f_x and f_y, f_x is said to call f_y *directly* if there is a call to f_y inside f_x.
- For any two functions f_x and f_y, f_x is said to call f_y *indirectly* if there is a call to another function f_{z_1} inside f_x, which in turn calls f_y, directly or indirectly.
- F_{d_1} be the subset of F_d containing functions that directly call one or more functions from F_i.
- F_{d_2} be the subset of F_d containing functions that indirectly call one or more functions from F_i.
- F_{i_1} be the subset of F_i containing functions that are called by one or more functions from F_{d_i}, directly of indirectly. $F_{i_2} = F_i - F_{i_1}$ (such that function in F_{i_2} can be placed absolutely anywhere in the program). In Figure 1,$F_d = \{f_1, f_2, f_3\}, F_i = \{f_4, f_5, f_6, f_7\}, F_{d_1} = \{f_1, f_2\}, F_{d_2} = \{f_3\}, F_{i_1} = \{f_4, f_5\}, F_{i_2} = \{f_6, f_7\}$.
- For any two functions f_x and f_y, $f_x < f_y$ represents the function f_x being defined before f_y.

There are two algorithms in the watermarking scheme, $insert(P, W, k) \rightarrow P_w$, $extract(P, P_w, k) \rightarrow W$, where P is the original program, W is the encrypted watermark, $W = \{w_1, \ldots, w_m\}$, $w_i \in \{0, 1\}$, P_w is the watermarked

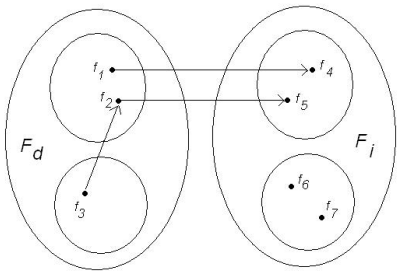

Fig. 1. Function interaction

program, and k is the key used to encrypt and decrypt the watermark.

For functions f_a, f_b in F_i, $f_a < f_b$ has the same meaning as $f_b < f_a$. However, we can encode the watermark in the sequence in which these functions are defined by inserting bogus function calls from f_a to f_b or from f_b to f_a, depending on the watermark bit. For example, if the watermark bit to be embedded is 0, we insert a call from f_a to f_b and therefore it is essential that in terms of sequencing, $f_b < f_a$ because $f_a < f_b$ will give a run-time error. We call this *creating a dependency* of f_a on f_b. Similarly, if watermark bit is 1, we can insert a call from f_b to f_a and place these functions such that $f_a < f_b$.

2.1. Watermark insertion

The set F_i of independent functions is selected from the original program P. $F_i = \{f_1, f_2, \ldots, f_n\}$ ($|F_i| = n, n \geq m$)and sorted based on code size (although other properties such as number of statements, number of variables used, return type, etc can be used to sort F_i). Let the sorted set be F', $F' = \{f'_1, f'_2, \ldots, f'_n\}$, $f'_i < f'_j$ if $i < j$. After the initial setup, we run Algorithm 1. The final ordering of F_i in the watermarked program is $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$.

2.2. Watermark extraction

The Company A (party who is sueing) produces unwatermarked code P and key k and the company B (party being sued) produces their version of the source code (which is watermarked according to A) in front of the Judge J. Extraction of watermark corresponding to A proves A's ownership. The extraction process has the following steps,

1. The set F_i is obtained from P and sorted according to the same criteria as in the insertion algorithm to get $F' = \{f'_1, f'_2, \ldots, f'_n\}$, $f'_i < f'_j$ if $i < j$.
2. Any forward declarations in the watermarked program P_w are replaced by corresponding function definitions. In P_w, F_i is $\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, $\sigma_i < \sigma_j$ if $i < j$.

$j = 1, C = (-1)^{w_a+1}$;
forall $a = 1$ *to* n *in steps of 1* **do**
  if $j = 1$ *AND* $C = -1$ **then**
    Call f'_1 in f'_n;
    Ignore f'_j for the rest of the embedding process;
    $j = n$ $\sigma_a = f'_1$;
  end
  if $j = n$ *AND* $C = +1$ **then**
    Call f'_n in f'_1;
    Ignore f'_j for the rest of the embedding process;
    $j = 1$ $\sigma_a = f'_n$;
  end
  if *(*$j \neq 1$ *OR* $C \neq -1$*) AND (*$j \neq n$ *OR* $C \neq +1$*)*
  then
    Call f'_j in f'_{j-C};
    Ignore f'_j for the rest of the embedding process;
    $j = j + C$ $\sigma_a = f'_{j-C}$;
  end
end

Algorithm 1: Watermark insertion algorithm

   (a) For $i = 1$ to n in steps of 1, swap σ_i with σ_n until no compile-time error is associated with σ_i.
   This step ensures that the sequencing of the functions is the same as the sequence imposed upon running the insertion algorithm.

3. Initialize $j = 1$, for $i = 1$ to n in steps of 1,
   if $\sigma_{i+1} = f'_{j+1}$, then $w_i = 1,\ j = j + 1$
   if $\sigma_{i+1} = f'_{j-1}$, then $w_i = 0,\ j = j - 1$

3. ANALYSIS

Re-ordering the functions results in a run-time error because of the dependency created. One way of getting around this problem from the attacker's perspective is to use function pre-declarations. The program would work correctly in this scenario but the extraction algorithm can still recover the watermark by moving the function definitions ahead of the "main" function, modifying the sequence till the program is successfully compiled, and then applying the recognition steps. This is computationally feasible for moderate sized watermarks (for example, up to 32 bits). Alternatively, the attacker needs to manually inspect the program to remove the bogus calls and then re-sequencing the functions. For a fairly large program, this is an tedious and inefficient attack. Furthermore, the attack can always remove a genuine function call which might result in abnormal execution of the program.

In additive attacks, the attacker adds certain function and bogus calls in the program such that the watermark-carrying-sequence is altered. Since our watermarking

Table 1. Execution time overhead introduced by watermarking scheme

Program	Execution time (in milliseconds		
	original	Watermarked with W_1	Watermarked with W_2
P_1	16.13	16.36	17.24
P_2	37.12	37.34	38.94
P_3	48.54	49.12	50.34
P_4	57.11	57.76	58.03
P_5	323.55	325.23	327.94

scheme is non-blind, the right set of functions can be extracted during recognition thereby surviving additive attacks too.

It is fairly evident that the capacity of the scheme is dependent on the number of function in F_i. The set F_i is permuted according to the values of watermark bits. For each watermark bit, one function is moved within F_i. Thus, a program with n functions in F_i can embed a watermark of size n bits.

To measure overheads introduced in execution time, we embedded 100-bit and 150-bit watermarks W_1 and W_2 respectively in 5 programs containing different number of functions (from 50 to 450) and the results are provided in Table 1. It indicates that the watermarking scheme introduces a small average overhead of 2.33% in execution time. The minimum overhead of 0.59% was introduced in program P_2 upon inserting W_2 and the maximum overhead of 6.88% in program P_1 when W_2 was inserted.

4. CONCLUSION

In this paper, we have presented a very simple, yet effective watermarking scheme for C/C++ source codes. The scheme introduces a negligible execution-time overhead (maximum slowdown during experiments was 6.88%). The watermark-carrying capacity is proportional to the number of functions in the program. The scheme is also resilient to re-sequencing, subtractive and additive attacks.

Using our algorithm, a company which suspects that a particular source code has been stolen from it can prove it's ownership over he source code in front of a judge. Building a blind watermarking scheme while maintaining security against additive attacks is our next project.

5. REFERENCES

[1] Christian Collberg, Edward Carter, Saumya Debray, Andrew Huntwork, Cullen Linn, and Mike Stepp. Dynamic path-based software watermarking. In *SIGPLAN '04 Conference on Programming Language Design and Implementation*, june 2004.

[2] Christian Collberg, Andrew Huntwork, Edward Carter, and Gregg Townsend. Graph theoretic software watermarks: Implementation, analysis, and attacks. In *Workshop on Information Hiding*, 2004.

[3] Christian Collberg, Stephen Kobourov, Edward Carter, and Clark Thomborson. Error-correcting graphs for software watermarking. In *Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science*, pages 156–167, 2003.

[4] Gaurav Gupta and Josef Pieprzyk. A low-cost attack on branch-based software watermarking schemes. In *Proceedings of International Workshop on Digital Watermarking*, volume 4283, pages 282–293, 2006.

[5] Gaurav Gupta and Josef Pieprzyk. Software watermarking resilient to debugging attacks. *Journal of Multimedia*, 2:10–16, 2007.

[6] Ginger Myles and Christian Collberg. Software watermarking through register allocation: Implementation, analysis, and attacks. In *International Conference on Information Security and Cryptology*, 2003.

[7] Ginger Myles and Hongxia Jin. Self-validating branch-based software watermarking. In *Proceedings of Information Hiding*, volume 3727, pages 342–356, 2005.

[8] Jasvir Nagra and Clark Thomborson. Threading software watermarks. In *Workshop on Information Hiding*, 2004.

[9] Gang Qu and Miodrag Potkonjak. Analysis of watermarking techniques for graph coloring problem. In *ICCAD*, pages 190–193, 1998.

[10] Gang Qu and Miodrag Potkonjak. Hiding signatures in graph coloring solutions. In *Information Hiding*, pages 348–367, 1999.

[11] Clark Thomborson, Jasvir Nagra, Ram Somaraju, and Charles He. Tamper-proofing software watermarks. In *Conferences in research and practice in information technology*, 2004.

[12] Ramarathnam Venkatesan, Vijay Vazirani, and Saurabh Sinha. A graph theoretic approach to software watermarking. *Lecture Notes in Computer Science*, 2137:157–168, 2001.