A Fragile Software Watermarking for Tamper-Proof

Changle Zhang, Hong Peng, Xianzhong Long, Zheng Pan, Ying Wu
School of Mathematics and Computer Engineering
Xihua University
Chengdu, China
changle_zhang@126.com, ph66@tom.com

Abstract

A fragile software watermarking scheme for integrity verification of software is proposed in this paper. The algorithm uses the idea of semantic-preserving code substitution for embedding the watermark. With the system, the generated watermark relates closely to the content of software, and the scheme has highly sensitive to different types of attacks. Furthermore, both watermark generating controlled by a key and watermark embedding position under the control of another key enhance the security of watermarking scheme. The scheme not only can effectively detect tampering, but also has the ability to identify the type of tampering clearly.

1. Introduction

Software watermarking [1]–[3] is a technique for embedding a secret message into a piece of software in order to encode some identifying information about it. When an unauthorized use of this software occurs, the secret message may be identified the ownership of the software. A robust software watermarking can be extracted even if it has been suffered malicious or casual semantics-preserving code transformation. Such watermarking is used in systems to protect the software ownership. A fragile software watermarking will (ideally) always be destroyed when the software has been changed. Such watermarking is used in authenticity and integrity verification of software.

With the development of software industries and the Internet, it is popular that the transmission and distribution of software via web. At this situation, more serious consequences would result from malicious attacks or virus damage in the process of software distribution. These applications lead an urgent demand that we carry out authenticity and integrity verification of software. Fragile watermarking is one of the effective ways to solve the problem mentioned above precisely, so we will focus on fragile watermarking.

2. Related Work

The first software watermarking algorithm was proposed by Davidson and Myhrvold [4]. It embeds a watermark into a program by reordering the basic blocks of the program. The register allocation algorithm was proposed by Qu and Potkonjak [5]. This method inserts a watermark into the interference graph of a program.

Applying the spread-spectrum watermarking method to software watermarking was introduced by Stern et al. [6]. Code is viewed not as a set of sequential instructions, but as a statistical object. What it really marks is the frequency counts of sets of consecutive instructions. In [8] Curran et al. also proposed a spread-spectrum software watermarking method which used call graph depth as a signal. An algorithm to insert watermark into a dummy method never intended for execution was proposed by Monden et al. [9]. This dummy method is guarded by an opaque predicate. Another algorithm to add watermarks into an opaque predicate or a dummy method was also introduced by Arboit [7].

The graph-based software watermarking scheme was proposed by Venkatesan et al. [10], which is to convert the software and the watermark code into digraphs and add new edges between the two graphs implemented by adding function calls between the software and watermark code. A threading software watermarking algorithm was introduced by Nagra and Thomborson [11], which the central idea is to encode the watermark in the sequence of the threads that are executed.

So far, most existing proposals for software watermarking have the feature that the mark is used for copyright protection. However, there is an urgent demand for integrity verification of software in practical applications, such as transmission and distribution. We will propose a code substitution-based fragile scheme for software watermarking in this paper. The scheme has several interesting consequences. First, encode watermark with the idea of semantic-preventing code substitution. Equivalent code has semantic meaning in itself as well as watermark meaning. Therefore, the encoding has a better level of information hidden. Second, with the system, use the encryption algorithm SHA-1 and pseudo-random function under the control of the key to generate watermark sequence which is closely to the content of software. Provide sensitive ability to detect tampering. Security of the algorithm can be enhanced by large key space. Third, not only does the scheme detect tampering effectively, but also it has the ability to identify the type of tampering.

978-0-7695-3744-3/09 $25.00 © 2009 IEEE
DOI 10.1109/IAS.2009.99
3. The Fragile Watermarking Scheme

A fragile watermarking scheme for the authenticity and integrity verification of software is proposed in this paper, in which the scheme is combined with the idea of the semantic-preserving code substitution. In the first place, we scan the original program and build a code book in which we divide the corresponding codes into two arrays. One array express the watermark bit “0”, the other express the watermark bit “1”. Set up the mapping relation between equivalent code and watermark bit. In the second place, we use pseudo-random function controlled by a key to generate a 0-1 sequence which is closely to the content of software. In order to embed the watermark, we replace the original code with equivalent code corresponding to watermark bit according to the mapping relation in the equivalent code substitution book (ECSB). In order to enhance the security, the watermark embedding positions are random under the control of a key. In the extraction phase, to judge effectively whether program has been tampered or not, we only compare the watermark regenerated from the content of program with the extracted one.

3.1. Principle of Code Substitution

There are many semantic equivalent codes or code groups in software. These codes or code groups will be used to achieve watermark embedding. Conveniently, we only use JAVA bytecode as an example in this paper. For instance, assume the existence of the following equivalent code group of JAVA bytecode:

Example 1: semantics $X \leftarrow X + Y$:

\[
\begin{align*}
\text{iload } X \\
\text{bipush } Y \\
\text{iinc } X \ Y & \quad \leftrightarrow \quad \text{idiv} \\
\text{istore } X 
\end{align*}
\]

These code groups are semantically equivalent. When it is replaced by a substituted code (group), the behavior of an original code (group) has not been changed. Accurately, both original code (group) and substituted code (group) have a same result in the run-time. Therefore, we look for all equivalent codes or code groups and build a code book. Assume the existence of $S$ groups, Table 1 is an example of code book that is composed of JAVA bytecode.

The main purpose of using equivalent code (group) is to embed the watermark. In order to do this, we need build the mapping relation between equivalent code (group) and watermark (0 or 1). Table 1 shows the equivalent code (group) is divided into two arrays, in which one of arrays corresponds to the watermark “0” and the other corresponds to the watermark “1”. For instance, the code “iinc X Y” in example 1 corresponds to the watermark bit “0”, and the code group “iload X; bipush Y; iadd; istore X” corresponds to the watermark bit “1”. In the embedding phase, if the watermark bit is “0”, we will choose code (group) of the first array to substitute original code (group), such as “iinc X Y”, otherwise, choose the other array to substitute. In the extraction phase, the code (group) in the first array expresses the watermark bit “0”, otherwise, the code (group) in the second array expresses the watermark bit “1”.

For achieving the relation mentioned above, we look for all equivalent code from the original program $P$ according to the code book, and then build the equivalent code substitution book (ECSB). To illustrate easily, the equivalent code belongs to the first array expresses respectively a, b, c, d, e, f,..., and the other expresses respectively A, B, C, D, E, F,..., Table 2 shows how to build the equivalent code substitution book (ECSB).

Table 1. code book

<table>
<thead>
<tr>
<th>No.</th>
<th>Original code (group)</th>
<th>Substituted code (group)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iload X</td>
<td>getstatic Y Z</td>
</tr>
<tr>
<td></td>
<td>getstatic Y Z</td>
<td>iadd</td>
</tr>
<tr>
<td>2</td>
<td>iconst 1</td>
<td>bipush X</td>
</tr>
<tr>
<td></td>
<td>istore X</td>
<td>idiv</td>
</tr>
<tr>
<td></td>
<td></td>
<td>istore X</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>iinc X Y</td>
<td>iload X</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bipush Y</td>
</tr>
<tr>
<td></td>
<td>iadd</td>
<td>istore X</td>
</tr>
</tbody>
</table>

Table 2. equivalent code substitution book (ECSB)

<table>
<thead>
<tr>
<th>No.</th>
<th>Watermark &quot;0&quot;</th>
<th>Watermark &quot;1&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>D</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>f</td>
<td>F</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

For achieving the relation mentioned above, we look for all equivalent code from the original program $P$ according to the code book, and then build the equivalent code substitution book (ECSB). To illustrate easily, the equivalent code belongs to the first array expresses respectively a, b, c, d, e, f,..., and the other expresses respectively A, B, C, D, E, F,..., Table 2 shows how to build the equivalent code substitution book (ECSB).

We observe that each equivalent code (group) in the ECSB has two kinds of meaning: one is semantic meaning in itself, the other is watermark meaning.

3.2. Watermark Generating

In order to achieve tampering detection for fragile watermarking, the generated watermark will be closely to the content of original program $P$ (binary code or byte code). Assume there are $N$ positions with the existence of equivalent code (group). These positions will be embedding positions in the embedding phase. The principle of generating is described as follows:

Firstly, according to the ECSB, we replace all the code (group) corresponding to “1” in original program $P$ with the code (group) corresponding to “0”. Therefore, we will obtain a new program $P_1$. 

310
Secondly, we use the encryption algorithm SHA-1 to convert the \( P_1 \) into a character string of 160 bits length (Assuming the private key is \( K_1 \)), and then get an accumulated value \( \text{sum} \) by accumulating the ASCII code of each character in the string.

Lastly, being \( \text{sum} \) as a seed, we use pseudo-random function to generate a 0-1 sequence. The component of pseudo-random function depends on a factor, which is the content of \( P_1 \). The factor will ensure that the generated watermark connects closely to the content of \( P_1 \). Thus, the generation of watermark can be represented by

\[
W = w_1, w_2, ..., w_n = \text{Random}(\text{sum}, K_1)
\]

where \( \text{Random} \) is a pseudo-random function, \( \text{sum} \) is an accumulated value, \( K_1 \) is a key.

3.3. Embedding

According to the principle of building ECSB, we know that an equivalent code (group) has a dual meaning, one is semantic meaning in itself, the other is watermark meaning. In the embedding phase, we don’t embed “0” or “1” itself into program \( P \). If the watermark to be embedded is “0”, we replace the code (group) embedding position with equivalent code (group) corresponding to “0”, otherwise, we choose the equivalent code (group) corresponding to “1”. It can be described in detail by example 1. Assume the current code is \( \text{“inc} X \text{ Y}” \), if the watermark to be embedded is “0”, we don’t carry out code substitution, otherwise, we will replace it with the code group “\text{ipush} X; \text{iadd}; \text{istore} X”.

There are \( N \) embedding positions for embedding watermark. We use 1, 2, ..., \( N \) to denote the corresponding embedding position. For the watermark sequence \( W = w_1, w_2, ..., w_n \), we will encode watermark according to the principle mentioned above. In order to enhance the security, we take a new position sequence \( i_1, i_2, ..., i_n \) using a pseudo-random function controlled by a key \( K_2 \), and then obtain a mapping relation between the watermarks and new embedding positions \( w_1 \leftrightarrow i_1, w_2 \leftrightarrow i_2, ..., w_j \leftrightarrow i_j, ..., w_n \leftrightarrow i_n \). After embedding all watermark bits, we obtain a new watermarked program \( P_w \).

3.4. Extraction

Extracting the watermark from the watermarked program \( P_w \), we will make use of the ECSB, \( K_1 \) and \( K_2 \), in which \( K_1 \) is used to generate watermark and \( K_2 \) is used to determine the embedding position. The principle of extracting is described as follows:

1. We replace all the code (group) corresponding to “1” in program \( P_w \) with the code (group) corresponding to “0” by exploiting the ECSB. Therefore, we will obtain a new program \( P_1’ \).

2. According to the method of watermark generating, we use \( K_1 \) and regenerate a new watermark \( W_1’ \) from \( P_1’ \).

3. Determine the position of equivalent code (embedding position) according to the ECSB and the relation between the extracted watermark and actual embedding position by using \( K_2’ \). Then extract watermark \( W_2’ \) according to the watermark meaning expressed by the equivalent code (group) in embedding position.

4. We compare the regenerated watermark \( W_1’ \) with the extracted one \( W_2’ \) in order to judge whether \( P_w \) is tampered or not. If they are completely identical, we can assert that the program \( P_w \) has not been tampered, otherwise, the program \( P_w \) has been tampered.

4. Performance Analysis

4.1. The Judgment of Tampering Type

The algorithm not only detects tampering effectively, but also has the ability to identify the type of tampering. From the types of tampering mentioned above, there are three principal kinds of attacks on watermarking systems.

a) Tamper with program content, but does not destroy the watermark.

b) Not only are program content tampered, but also the watermark are destructed.

c) Tamper with program content, at the same time, generate a pseudo-watermark and embed it into the program.

For the first case, because the content of a program is tampered and the watermark is unchanged, according to the mapping relation of ECSB and the current program content, the regenerated watermark is certain to be different with the extracted watermark from the program. The algorithm can accurately identify this circumstance.

For the second one, the embedding position of watermark is randomly generated by the key \( K_2 \), in which \( K_2 \) is to be kept secret from an attacker, so he does not determine the watermark embedding position. Even if the attacker can destroy the watermark, he is not aware which part of the code corresponds to the watermark. Thus, we can easily determine which code has been tampered and judge which watermark corresponding to the code has been destroyed.

In the last case, because he doesn’t clear the \( K_1 \), even if the attacker knows the content of ECSB, the counterfeit watermark is different with the legal one. Besides, even if the attacker can successfully forge watermark, the embedding position of the counterfeit watermark is not right because he doesn’t have the \( K_2 \) which is used to control the embedding position. Therefore, we can easily know which the content is tampered through comparing the regenerated watermark.
with the extracted one, meanwhile, we can also judge which
tampering position, security is stronger. Moreover, it has a good tampering detection and the ability
of stealth; it is very sensitive to all kinds of tampering, even if the watermarked program has been changed by
these attacks, the algorithm can easily detect whether the
watermarked program is tampered or not. The security of
the algorithm is guaranteed.

4.3. Stealth

Comparing with the traditional scheme such as digital
signature which is used integrity verification, the software
watermarking scheme has a high level of stealth and doesn’t
need extra mechanism to save additional signature informa-
tion. More importantly, that most existing schemes for soft-
ware watermarking add static data or extra confusing code
in program would result in attackers’ curiosity and malicious
attack. So we encode watermark with the idea of semantic-
preventing code substitution in this paper. Therefore, the
stealth of the algorithm is also guaranteed.

4.4. Data Rate

Data rate is the amount of embedding watermark in
code and it is an important criterion to evaluate watermark
algorithm. The number of bits that can be encoded depends
on the number of substituted codes, and the number of
substituted codes depends on the program scale. Hence, data
rate is proportional to the program scale; and the larger the
program scale is, the larger the data rate is. Vise versa.

5. Conclusion

A fragile watermarking scheme for Tamper-Proof of
software is proposed in this paper. The scheme has the
following characteristics: use the encryption algorithm SHA-
1 and pseudo-random function under the control of the key
K1 to generate watermark. And the embedding position of watermark is randomly generated by
the key K2. Because both K1 and K2 have a large key
space, the attacker cannot carry out effective attacks, and tiny
tampering can be detected in the extraction phase. Therefore,
even if the watermarked program has been changed by
these attacks, the algorithm can easily detect whether the
watermarked program is tampered or not. The security of
the algorithm is stronger.

Acknowledgment

This work is supported by the importance project foun-
dation of the education department of Sichuan province
(No.2007ZA112), the training fund of science and tech-
nology of Sichuan province (No.08209057), China, and
the importance project foundation of Xihua University
(No.ZG0722603).

References
[1] C. Collberg and C. Thomborson, Watermarking, tamper-
proofing, and obfuscation - tools for software protection, IEEE
Transactions on Software Engineering, vol.28, 2002, pp: 735-
746.
marking, ACM SIGPLAN Notices, Proceedings of the ACM
SIGPLAN 2004 conference on Programming language design
and implementation, vol.39, Iss. 6, June 2004.
A general architecture for software watermarking, Technical
[4] R. Davidson and N. Myhrvold, Method and system for gen-
erating and auditing a signature for a computer program, US Patent 5,559,884, September 1996. Assignee: Microsoft
Corporation.
for Graph Coloring Problem, Proceeding of 1998 IEEE/ACM
International Conference on Computer Aided Design, ACM
Press, 1998, pp. 190-193
Opaque Predicates, The Fifth International Conference on
Electronic Commerce Research (ICECR-5), 2002
Software Watermarking, PPPJ 2003, Kilkenny City, Irenland,
pp. 145-148
A Practical Method for Watermarking Java Programs,
The 24th Computer Software and Applications Conference (comp-
Approach to software watermarking, presented at 4th Inter-
national Information Hiding Workshop, Pittsburgh, PA, USA,
Apr. 2001
marks, Proc. Information Hiding Workshop, 2004