A Survey of Static Software Watermarking

James Hamilton and Sebastian Danicic
Department of Computing, Goldsmiths, University of London
United Kingdom
james.hamilton@gold.ac.uk, s.danicic@gold.ac.uk

Abstract

Software watermarks, which can be used to identify the intellectual property owner of a piece of software, are broadly divided into two categories: static and dynamic. Static watermarks are embedded in the code and/or data of a computer program, whereas dynamic watermarking techniques store a watermark in a program’s execution state. In this paper, we present a survey of the known static software watermarking techniques, including a brief explanation of each.

1 Introduction

The global revenue loss due to software piracy was estimated to be more than $50 billion in 2009 [8]. Software companies regularly use legal methods such as copyright laws, patents and license agreements and ethical arguments such as fair compensation for producers. However, these methods do not always dissuade people from stealing software, especially in emerging markets where the price of software is high and incomes are low [27].

Software watermarking involves embedding a unique identifier within a piece of software, to discourage software piracy. Watermarking does not prevent copying but instead discourages software thieves by providing a means to identify the owner of a piece of software and/or the origin of the stolen software [54]. The hidden watermark can be recognised or extracted, at a later date, by the use of a recogniser or extractor to prove ownership of stolen software [86]. It is also possible to embed a unique customer identifier in each copy of the software distributed which allows the software company to identify the individual that pirated the software [58].

Watermarking techniques are used extensively in the entertainment industry to identify multimedia files such as audio and video files, and the concept has extended into the software industry. Software watermarking is one technique available for the protection of software [59].

Watermarks can be classified as either static or dynamic: static watermarks are embedded in the code and/or data of a computer program, whereas dynamic watermarking techniques store a watermark in a program’s execution state [15]. Figure 1 shows a conceptual diagram of a simple static watermarking system.

![Figure 1. A simple static watermarking system](image)

In this paper, we present a survey of different types of static software watermarking techniques, including a brief explanation of each of the known algorithms.

2 Code Replacement

Some of the first patented software watermarking algorithms [43, 69] were based around the idea of code replacement; that is, they replaced a pre-determined portion of code and/or data in a program with the watermark value. These early techniques were susceptible to attacks, such as collusion attacks – where the attacker compares two or more copies of a watermarked program to identify the location of the watermark. Monden et al. [51, 52, 53] describe a technique, MON, for watermarking Java programs by swapping bytecode instructions within dummy methods (implemented as jmark [50]). The dummy methods used by MON are created either manually or automatically, and method calls are protected by opaque predicates [15] to ensure they are not executed. The basic idea is to assign bit values to certain Java bytecode instructions and replace the existing instructions with the encoding bits which correspond to the watermark value. As the dummy method is not executed there is no semantic restrictions on the re-
placements but the watermarked method must be semantically correct, in order to pass the Java bytecode verifier \cite{12}. Myles et al. \cite{57} implemented a version, MON$_{SM}$, in Sandmark \cite{12} and evaluated compared it to the Davidson/Myhrvold watermarking scheme \cite{29}. MON$_{SM}$ differs from the jmark implementation as it automatically generates a dummy method, so is completely automatic. However, it is difficult to generate code which is similar to the original program and it may be discoverable by a statistical analysis of the bytecode.

Fukushima and Sakurai \cite{33}, Fukushima et al. \cite{34} combine the MON watermarking technique with obfuscation to provide protection against collusion attacks. The idea is to obfuscate each copy of a program differently, so that comparison of programs will show several differences – not just the watermark location.

Pervez et al. \cite{61} describe a MON-like system which acts on Java source, instead of bytecode and Thaker \cite{75} introduces a similar scheme which embeds watermarks by swapping semantically-equivalent x86 assembly instructions. Another tool, Hydan \cite{31,32,30} similarly uses the replacement of semantically-equivalent instructions for the purpose of steganographically embedding secret information in x86 binary programs. Anckaert et al. \cite{4,5} further discuss Hydan’s technique for steganography, comparing it to other techniques for hiding information in computer programs.

Stern et al. \cite{74} introduce robust object watermarking ROW, based on a spread-spectrum technique previously used for multimedia watermarking \cite{26}. This technique differs from many other techniques because it views the code as a whole statistical object, rather than a sequence of instructions. The technique is more resilient against collusion attacks because the watermark is spread out over the program, rather than being in one location.

The approach modifies the frequencies of groups of instructions in order to watermark the code (though other statistical properties of the program could be used). Stern et al. \cite{74} implemented their technique for x86 assembly language and later Hachez \cite{59}, and separately Collberg and Sahoo \cite{14}, implemented the technique for Java bytecode. Curran et al. \cite{28} describe a spread-spectrum technique using a vector derived from the call graph depth of a program; Ai et al. \cite{3} attempt to improve on the original algorithm by introducing a collusion-attack resistant variation.

Finding the perfect transformation is difficult and none of the existing implementations are perfect as many of the transformations are easily undoable by trivial obfuscations \cite{13}.

3 Code Re-Ordering

Davidson and Myhrvold \cite{29} proposed one of the first software watermarking algorithms DM which encodes the watermark by basic block re-ordering. Myles et al. \cite{57} proposed a method of watermark extraction for this technique and implemented the DM algorithm in Sandmark \cite{12}.

The basic idea is to convert the watermark into a number \(w\); then the \(w^{th}\) permutation of a set of basic blocks \(B\) is generated. The permuted basic blocks \(B'\) are re-linked to retain the original program semantics and \(B\) is replaced by \(B'\) to produce the watermarked program \(P'\). To extract a watermark, the ordering of the original basic blocks is compared against the new ordering, to obtain the permutation number; this number is then converted back into the watermark number.

Hattanda and Ichikawa \cite{41} evaluated the DM watermarking algorithm by watermarking several C programs and analyzing metrics such as program size and program performance. In their implementation they found that the size increase of a watermarked program was between 9% and 24% while the performance was 86% to 102% of the original program. Anckaert et al. \cite{4,5} implemented and evaluated a version of the DM watermarking algorithm for machine code where groups of chains of basic blocks are reordered. They concluded that their watermarking algorithm is stealthier as it has a minimal affect on code locality.

Shirali-Shahreza and Shirali-Shahreza \cite{71} proposed a software watermark scheme based on the re-ordering of operations in mathematical equations. The idea involves re-ordering symmetric mathematical operations, such as addition, to preserve program semantics. Sha et al. \cite{70} proposed a very similar technique based on re-ordering operand coefficients.

Gong et al. \cite{37} proposed a watermarking scheme for Java based on the ordering of a class file’s constant pool. The constant pool, of a Java classfile, is an array of variable length elements containing every constant used in the Java class \cite{78}. This scheme involves re-ordering constants corresponding to the \(w^{th}\) permutation of the constants where \(w\) is an integer watermark.

4 Register Allocation Based Watermarking

The QP algorithm \cite{64} is a constraint-based watermarking (and fingerprinting \cite{66}) algorithm based on the concept of graph colouring. In the QP algorithm edges are added to a graph based on the value of the watermark. The graph used for watermarking programs is the interference graph, which is used to model the relationship between the variables in a program method. Each vertex in the graph represents a variable and an edge between two variables indicates that their live ranges overlap. We colour the graph in order to minimise the number of registers required and ensure that two live variables do not share a register. Zhu and Thomborson \cite{82} described a clarified version of the originally published algorithm.

A major flaw in the QP algorithm is that it is not ex-
tractable as it is possible to insert two different messages into an interference graph and obtain the same watermark graph \cite{84, 83, 85}. It has also been shown that the QP graph solution can be modified in such a way that any message could be extracted \cite{46}. Qu and Potkonjak dismiss this problem, claiming that it will be hard to build a meaningful message particularly if the original message is encrypted by a one-way function \cite{65}.

Myles and Collberg \cite{55} implemented a new algorithm, QPS, in Sandmark \cite{12}. In the QPS algorithm triples of vertices are selected such that they are isolated units that will not effect other vertices in the graph. Experimental results \cite{55} showed that the QPS algorithm has a very low data-rate and is susceptible to a variety of simple attacks, such as obfuscations. However, the QPS algorithm was found to be quite stealthy and is extremely credible. In other words, the watermarks are hard to detect by an attacker whilst readily detectable by the watermark author.

Zhu and Thomboro\cite{83} proposed a further improvement which they call the QPI algorithm. The QPI extraction algorithm requires the original interference graph and the watermarked graph in order to extract the watermark message. The coloring of candidate vertices from the original graph and the watermarked graph are compared, to extract the watermark.

The Colour Change (CC) and Colour Permutation (CP) algorithms \cite{47, 48} are further improvements on the QPS algorithm where the colouring function is modified to embed a message, rather than modifying the interference graph. The CP algorithm converts the watermark bit string into a natural number \( w \), and then chooses the \( w \)-th permutation of the lexicographically ordered colours to replace the original colour. The data-rate of the CC and CP algorithms is higher than that of QPS and QPI because each vertex in the interference graph can store 1 watermark bit. Li and Liu \cite{49} proposed a more efficient algorithm based on the CC algorithm which they called Selected Colour Change (SCC). The efficiency increase is obtained by only changing the colours of either the 1 or the 0 bits, but not both.

Jiang et al. \cite{44} propose a technique based on a combination of RSA public-key encryption \cite{68} and the QPI algorithm \cite{83}. Simply, they suggest that the watermark bit string is encrypted before being embedded. If an adversary extracts the encrypted watermark they will not be able to decipher it, if the encryption is strong enough.

5 Graph Watermarking

Graph watermarking algorithms rely on the fact that graph-generating code is difficult to analyse due to aliasing effects \cite{35} which, in general, is known to be undecidable \cite{67}. Collberg et al. \cite{19} describe several techniques for encoding watermark integers in graph structures. The graph encodings can be divided into 3 types of encoding: enumeration, radix and permutation.

Enumeration encoding encodes the watermark number \( w \) as the \( w \)-th enumerated graph, of a specific graph family. These families include: directed Parent-Pointer Trees, which contain a single edge between a vertex and its parent, and Planted Planar Cubic Trees which are binary trees where every interior vertex, except the root, has two children.

A parent-pointer tree data-structure is space efficient because each node has just one pointer field referencing its parent. However, the data-structure is fragile and an adversary could add a single node or edge to distort the watermark. PPCTs can be made more resilient to attacks by a) marking each leaf with a self-pointer, and b) creating an outer cycle from the root to itself through all the leaves \cite{13}.

Radix graphs add an extra pointer field in each vertex of a circular linked list of length \( k \) to encode a base-\( k \) digit. It is possible to encode watermark digits where a self-pointer represents 0, a pointer to the next node 1, and so on.

Several papers \cite{9, 62, 81, 79, 45, 82} describe a technique which combines radix graphs with the error-correcting properties of PPCTs by converting a radix graph into a PPCT-like structure.

Permutation based graphs (as defined by Collberg et al. \cite{19}) use the same basic singly linked circular list structure as the radix graphs but have error-correcting properties. In this encoding scheme a permutation \( P = \{p_1, p_2, \ldots, p_n\} \) is derived from the watermark integer; the permutation is then encoded in the graph by adding edges between vertices \( i \) and \( p_i \).

Reducible permutation graphs (RPG) \cite{77, 76} are very similar to permutation graphs but they closely resemble control-flow graphs as they are reducible-flow graphs \cite{42}. RPGs, like CFGs, contain a unique entry node and a unique exit node, a preamble which contains zero or more nodes from which all other nodes can be reached and a body which encodes a watermarking using a self-inverting permutation \cite{10}. This family of graphs is resistant to edge-flip attacks, where an attacker inverts the condition of conditional jumps in a program.

Venkatesan et al. \cite{77} proposed the first static graph watermarking scheme, Graph Theoretic Watermarking (GTW), which encodes a value in the topology of a program’s control-flow graph \cite{2}. The idea was later patented by Venkatesan and Vazirani \cite{76} for Microsoft. The basic concept is to encode a watermark value in a reducible permutation graph and convert it into a control flow graph; it is then merged with the program control flow graph by adding control flow edges between the two.

The algorithm adds bogus control flow edges between random pairs of vertices in the program CFG and watermark CFG in order to protect against static analysis attacks looking for sparse-cuts \cite{7} in the control-flow graph. A sparse-
cut would indicate a possible joining point of the original program CFG and the watermark CFG where the attacker could split the program with as few edges broken as possible.

Collberg et al. [11] implemented a version GTWSM of GTW in Sandmark [12]. They measured the size and time overhead of watermarking and evaluated the algorithm against a variety of attacks. They also introduce two methods (Partial Sum splitting and Generalised Chinese Remainder Theorem splitting) for splitting a watermark integer into redundant pieces so that a large integer can be stored in several smaller CFGs. They found that stealth is a big problem; for example, the basic blocks of the generated watermark method consisted of 20% arithmetic instructions compared to just 1% for standard Java methods [21]. Watermarks of up to 150 bits increased program size by between 40% and 75%, while performance decreased by between 0% and 36% [11].

6 Opaque Predicates

An opaque predicate is a predicate whose outcome is known a priori. It is difficult for automated software analysis to find the value of the predicate; therefore it is not known whether the enclosed code (which may or may not be a watermark) could be removed [16].

Arboit [6] proposed a watermarking method where pieces of a watermark are encoded as constants within opaque predicates. The watermark is extracted by searching a program for the watermark opaque predicates and decoding them back into the watermark value.

Myles and Collberg [56] implemented the algorithm in Sandmark [12] and found that the algorithm could, fairly easily, be defeated by semantics-preserving transformation attacks.

7 Abstract Interpretation

Abstract interpretation [24] is a static analysis technique used for, among other things, the verification of software. The technique is used to prove that the abstract semantics of a program satisfies an abstract specification [23], ignoring irrelevant details about the concrete semantics and specifications. Abstract interpretation can answer questions which do not need full knowledge of program executions or which tolerate an imprecise answer, such as partial correctness proofs of programs [24].

The basic idea is to hide the watermark in such a way that it can only be extracted by an abstract interpretation of the concrete semantics of the code [25]. A constant propagation static analysis [2] is used to extract a watermark embedded using the abstract interpretation. During normal execution of a program the variable takes on several values, however, during an abstract interpretation of the program the variable reveals the watermark value.

An advantage of this technique is that the code is actually executed and therefore there is no need to use opaque predicates, to prevent dead-code removal [2]. However, the code generated is not entirely stealthy, as the watermark variable can take on uncommon values during execution [13]. Giacobazzi [36] discusses abstract interpretation based techniques for designing new code obfuscation and watermarking techniques.

Preda et al. [63] built a prototype tool which is able to detect the opaque true predicates \( \forall x \in \mathbb{Z} : 2(x + x) \) and \( \forall x \in \mathbb{Z} : 2(x + x) \) by executing the code in the \( n \)-arity abstract domain. In this domain the variables take on the ‘value’ \( \text{odd or even} \). Abstract operations are performed on the abstract values, such as addition where, for example, \( \text{odd} + \text{odd} = \text{even} \).

8 Watermarking Systems

There are 4 widely available watermarking systems for Java bytecode: Sandmark [12], Allatori [72], DashO [1] and jmark [50]. SandMark [12] is a tool developed by Collberg et al. at the University of Arizona for research into software watermarking, tamper-proofing, and obfuscation of Java bytecode. Sandmark, which was last updated in 2004, includes 12 static software watermarking algorithms [18] – many of which are implementations of algorithms discussed in this paper (including jmark’s algorithm). Allatori [22] is a commercial Java obfuscator which also includes a watermarking system. DashO [1] is a commercial Java security solution, including obfuscator, watermarker and encrypter. The static watermarking algorithms in all of these systems are susceptible to semantics-preserving transformation (distortive) attacks.

UWStego [17] is a tool for developing watermarking algorithms that was never generally released, unless requested, and is no longer under development. Additionally, JavaWiz [60], a software watermarking system, is no longer available.

Hydan [30], a system for steganographically embedding hidden messages in x86 assembly code, is available but is not aimed at watermarking and is therefore not resilient against attacks.

9 Conclusion

We have presented a survey of static software watermarking schemes from the basic, early patents [43] to the latest register allocation based techniques [49]. Previous studies (e.g. [41] [11] [14] [11]) have shown that static techniques are highly susceptible to semantics preserving transformation attacks and are therefore easily removed by an adversary.

Software watermarking can be supplemented with other forms of protection [73], such as obfuscations or tamper-proofing techniques [22], in order to better protect a pro-
gram from copyright infringement and decompilation [39]. Further research should focus on dynamic software watermarking algorithms [80] as static watermarking schemes are not robust enough for intellectual property protection.

References

[21] Christian Collberg, Andrew Huntwork, Edward...


