A survey of static software watermarking

Conference Paper · March 2011

DOI: 10.1109/WorldCIS17046.2011.5749891 · Source: IEEE Xplore

Sebastian Danicic
Goldsmiths, University of London

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 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.

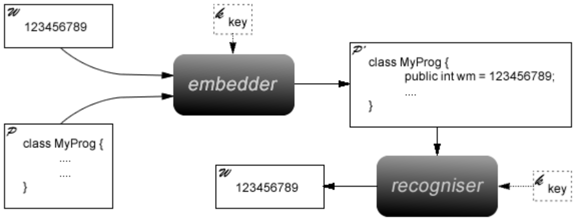

Figure 1. A simple static watermarking system

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 [16] 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 replacements but the watermarked method must be semantically correct, in order to pass the Java bytecode verifier [78]. Myles et al. [57] implemented a version, MON_{SM}, in Sandmark [12] and evaluated compared it to the Davidson/Myhrvold watermarking scheme [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 [33], Fukushima et al. [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. [61] describe a MON-like system which acts on Java source, instead of bytecode and Thaker [75] introduces a similar scheme which embeds watermarks by swapping semantically-equivalent x86 assembly instructions. Another tool, Hydan [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. [4, 5] further discuss Hydan's technique for steganography, comparing it to other techniques for hiding information in computer programs.

Stern et al. [74] introduce robust object watermarking ROW, based on a spread-spectrum technique previously used for multimedia watermarking [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. [74] implemented their technique for x86 assembly language and later Hachez [38], and separately Collberg and Sahoo [14], implemented the technique for Java bytecode. Curran et al. [28] describe a spread-spectrum technique using a vector derived from the call graph depth of a program; Ai et al. [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 [13].

3 Code Re-Ordering

Davidson and Myhrvold [29] proposed one of the first software watermarking algorithms DM which encodes the watermark by basic block re-ordering. Myles et al. [57] proposed a method of watermark extraction for this technique and implemented the DM algorithm in Sandmark [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 permutated 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 [41] evaluated the DM watermarking algorithm by watermarking several C programs and analysing 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. [4] 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 [71] proposed a software watermark scheme based the re-ordering of operations in mathematical equations. The idea involves reordering symmetric mathematical operations, such as addition, to preserve program semantics. Sha et al. [70] proposed a very similar technique based on re-ordering operand coefficients.

Gong et al. [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 [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 [64] is a constraint-based watermarking (and fingerprinting [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 [84] 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 [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 [46]. Qu and Potkonjak dismiss this problem, claiming that it will be hard to build a meaningful message particularily if the original message is encrypted by a one-way function [65].

Myles and Collberg [55] implemented a new algorithm, QPS, in Sandmark [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 [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 Thomborson [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 [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 [49] proposed a more efficient algorithm based on the CC algorithm which they call 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. [44] propose a technique based on a combination of RSA public-key encryption [68] and the QPI algorithm [83]. Simply, they suggest that the watermark bit string is encrypted before being embedded. If an adversery 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 [35] which, in general, is known to be un-decidable [67]. Collberg et al. [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 it's 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 it's 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 [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 [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. [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) [77, 76] are very similar to permutation graphs but they closely resemble control-flow graphs as they are reducible-flow graphs [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 [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. [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 [2]. The idea was later patented by Venkatesan and Vazirani [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 [7] in the control-flow graph. A sparsecut 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 GTW_{SM} 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 as 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 opaquely true predicates $\forall x \in \mathbb{Z} : 2|(x^2 + x)$ 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' *odd* or *even*. Abstract operations are performed on the abstract values, such as addition where, for example, *odd* $+_a$ *odd* = *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. [20] 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 [72] 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 [40].

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, 69] 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 program 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

[1] DashO, 2010. URL `http://www.preemptive.com/products/dasho/overview`. Accessed: 2 April, 2010.

[2] Alfred V Aho, Monica S Lam, Ravi Sethi, and Jeffrey D Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2nd edition, August 2006. ISBN 0321486811.

[3] Jieqing Ai, Xingming Sun, Yunhao Liu, Ingemar J. Cox, Guang Sun, and Yi Luo. A stern-based Collusion-Secure software watermarking algorithm and its implementation. In *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering*, pages 813–818. IEEE Computer Society, 2007. ISBN 0-7695-2777-9.

[4] Bertrand Anckaert, Bjorn De Sutter, and Koen De Bosschere. Covert communication through executables. In *Program Acceleration through Application and Architecture Driven Code Transformations: Symposium Proceedings*, pages 83–85, 2004.

[5] Bertrand Anckaert, Bjorn De Sutter, and Koen De Bosschere. Steganography for executables. *Information Security and Cryptology - ICISC 2004*, pages 425–439, 2005.

[6] Genevieve Arboit. A method for watermarking java programs via opaque predicates. In *The Fifth International Conference on Electronic Commerce Research (ICECR-5)*, 2002.

[7] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 222–231, Chicago, IL, USA, 2004. ACM. ISBN 1-58113-852-0.

[8] Business Software Alliance. Piracy impact study 2010: The economic benefits of reducing software piracy, 2010.

[9] XiaoJiang Chen, DingYi Fang, JingBo Shen, Feng Chen, WenBo Wang, and Lu He. A dynamic graph watermark scheme of tamper resistance. In *Proceedings of the 2009 Fifth International Conference on Information Assurance and Security - Volume 01*, pages 3–6. IEEE Computer Society, 2009. ISBN 978-0-7695-3744-3.

[10] Maria Chroni and Stavros D. Nikolopoulos. Encoding watermark integers as self-inverting permutations. In *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*, pages 125–130, Sofia, Bulgaria, 2010. ACM. ISBN 978-1-4503-0243-2.

[11] C. Collberg, A. Huntwork, E. Carter, and G. Townsend. Graph theoretic software watermarks: Implementation, analysis, and attacks. In *Workshop on Information Hiding*, 2004.

[12] Christian Collberg. Sandmark, August 2004. URL `http://www.cs.arizona.edu/sandmark/`. Accessed: 2 April, 2010.

[13] Christian Collberg and Jasvir Nagra. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley Professional, 2009. ISBN 0321549252, 9780321549259.

[14] Christian Collberg and Tapas Ranjan Sahoo. Software watermarking in the frequency domain: implementation, analysis, and attacks. *J. Comput. Secur.*, 13(5): 721–755, 2005.

[15] Christian Collberg and Clark Thomborson. On the limits of software watermarking. Technical Report 164, August 1998.

[16] Christian Collberg, Clark Thomborson, and Douglas Low. Manufacturing cheap, resilient, and stealthy opaque constructs. In *Principles of Programming Languages 1998, POPL'98*, January 1998.

[17] Christian Collberg, S. Jha, D. Thomko, and H. Wang. UWStego, 2001. URL `http://pages.cs.wisc.edu/~hbwang/watermark/`. Accessed: 14 July, 2010.

[18] Christian Collberg, Miriam Miklofsky, Ginger Myles, Ashok Purushotham, RathnaPrabhu Rajendran, Andrew Huntwork, Xiangyu Zhang, Danny Mandel, Anna Segurson, Martin Stepp, Kelly Heffner, J Nagra, G Townsend, Balamurugan Chirtsabesan, and Tapas Ranjan Sahoo. Sandmark algorithms. Sandmark documentation, University of Arizona, July 2002.

[19] 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.

[20] Christian Collberg, Ginger Myles, and Andrew Huntwork. Sandmark – A tool for software protection research. *IEEE Security and Privacy*, 1(04):4049, 2003. ISSN 1540-7993.

[21] Christian Collberg, Andrew Huntwork, Edward

Carter, Gregg Townsend, and Michael Stepp. More on graph theoretic software watermarks: Implementation, analysis, and attacks. *Inf. Softw. Technol.*, 51(1): 56–67, 2009.

[22] Christian S. Collberg and Clark Thomborson. Watermarking, Tamper-Proofing, and obfuscation - tools for software protection. In *IEEE Transactions on Software Engineering*, volume 28, page 735746, August 2002.

[23] P Cousot. Abstract interpretation. Online: `http://www.di.ens.fr/~cousot/AI/`, August 2008. Accessed: 15 September, 2010.

[24] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, page 238252, Los Angeles, California, 1977. ACM Press, New York, NY.

[25] Patrick Cousot and Radhia Cousot. An abstract interpretation-based framework for software watermarking. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 173–185, Venice, Italy, 2004. ACM. ISBN 1-58113-729-X.

[26] Ingemar J. Cox, Joe Kilian, Frank Thomson Leighton, and Talal Shamoon. A secure, robust watermark for multimedia. In *Proceedings of the First International Workshop on Information Hiding*, pages 185–206. Springer-Verlag, 1996. ISBN 3-540-61996-8.

[27] Gareth Cronin. A taxonomy of methods for software piracy prevention. Technical report, Department of Computer Science, University of Auckland, New Zealand, 2002.

[28] D. Curran, N.J. Hurley, and M. O. Cinneide. Securing java through software watermarking. In *Proceedings of the 2nd international conference on Principles and practice of programming in Java*, page 311324, 2003.

[29] Robert Davidson and Nathan Myhrvold. Method and system for generating and auditing a signature for a computer program, June 1996. Microsoft Corporation, US Patent 5559884.

[30] Rakan El-Khalil. Hydan. `http://www.crazyboy.com/hydan/`, 2004. Accessed: 11 September, 2010.

[31] Rakan El-khalil and Angelos D. Keromytis. Hydan: Hiding information in program binaries. In *in International Conf. on Information and Communications Security, ICICS*. Springer-Verlag, 2004.

[32] Rakan El-Khalil and Angelos D. Keromytis. Hydan: Information hiding in program binaries. In *International Conference on Informaton and Communications Security*, 2004.

[33] Kazuhide Fukushima and Kouichi Sakurai. A software fingerprinting scheme for java using classfiles obfuscation, 2004.

[34] Kazuhide Fukushima, Toshihiro Tabata, Toshiaka Tanaka, and Kouichi Sakurai. A software fingerprinting scheme for java using class structure transformation. *Transactions of Information Processing Society of Japan*, 46(8):2042–2052, August 2005. ISSN 03875806.

[35] Rakesh Ghiya and Laurie J. Hendren. Is it a tree, a DAG, or a cyclic graph? a shape analysis for heap-directed pointers in c. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–15, St. Petersburg Beach, Florida, United States, 1996. ACM. ISBN 0-89791-769-3.

[36] Roberto Giacobazzi. Hiding information in completeness holes: New perspectives in code obfuscation and watermarking. In *Proceedings of the 2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 7–18. IEEE Computer Society, 2008. ISBN 978-0-7695-3437-4.

[37] Daofu Gong, Fenlin Liu, Bin Lu, Ping Wang, and Lan Ding. Hiding information in java class file. In *Proceedings of the 2008 International Symposium on Computer Science and Computational Technology - Volume 02*, pages 160–164. IEEE Computer Society, 2008. ISBN 978-0-7695-3498-5.

[38] Gael Hachez. *A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards*. PhD thesis, Universite Catholique de Louvain, March 2003.

[39] James Hamilton and Sebastian Danicic. An evaluation of current java bytecode decompilers. In *Ninth IEEE International Workshop on Source Code Analysis and Manipulation*, volume 0, pages 129–136, Edmonton, Alberta, Canada, 2009. IEEE Computer Society.

[40] James Hamilton and Sebastian Danicic. An evaluation of static java bytecode watermarking. In *Lecture Notes in Engineering and Computer Science: Proceedings of The World Congress on Engineering and Computer Science 2010*, volume 1, pages 1 – 8, San Francisco, USA, October 2010. ISBN 978-988-17012-0-6. Winner of the Best Student Paper Award.

[41] Kazuhiro Hattanda and Shuichi Ichikawa. The evaluation of davidsons digital signature scheme. *IEICE Trans. Fundamentals*, E87A(1), January 2004.

[42] Matthew S. Hecht and Jeffrey D. Ullman. Flow graph

reducibility. In *Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 238–250, Denver, Colorado, United States, 1972. ACM.

[43] Keith Holmes. Computer software protection, February 1994. International Business Machines Corporation, US Patent: 5287407.

[44] Zetao Jiang, Rubing Zhong, and Bina Zheng. A software watermarking method based on Public-Key cryptography and graph coloring. In *Genetic and Evolutionary Computing, 2009. WGEC '09. 3rd International Conference on*, pages 433–437, 2009.

[45] Zhu Jianqi, Liu YanHeng, and Yin KeXin. A novel dynamic graph software watermark scheme. In *Proceedings of the 2009 First International Workshop on Education Technology and Computer Science - Volume 03*, pages 775–780. IEEE Computer Society, 2009. ISBN 978-0-7695-3557-9.

[46] Tri Van Le and Yvo Desmedt. Cryptanalysis of UCLA watermarking schemes for intellectual property protection. In *Revised Papers from the 5th International Workshop on Information Hiding*, pages 213–225. Springer-Verlag, 2003. ISBN 3-540-00421-1.

[47] Hakun Lee and Keiichi Kaneko. New approaches for software watermarking by register allocation. In *Proceedings of the 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 63–68. IEEE Computer Society, 2008. ISBN 978-0-7695-3263-9.

[48] Hakun Lee and Keiichi Kaneko. Two new algorithms for software watermarking by register allocation and their empirical evaluation. In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, pages 217–222. IEEE Computer Society, 2009. ISBN 978-0-7695-3596-8.

[49] Jun Li and Quan Liu. Design of a software watermarking algorithm based on register allocation. In *e-Business and Information System Security (EBISS), 2010 2nd International Conference on*, pages 1–4, 2010.

[50] Akito Monden. jmark, 2003. URL http://se.aist-nara.ac.jp/jmark/. Accessed: 14 July, 2010.

[51] Akito Monden, Hajimu Iida, et al. A watermarking method for computer programs. In *Proceedings of the 1998 Symposium on Cryptography and Information Security, SCIS'98*. Institute of Electronics, Information and Communication Engineers, January 1998. in Japanese.

[52] Akito Monden, Hajimu Iida, Ken ichi Matsumoto, Katsuro Inoue, and Koiji Torii. Watermarking java programs. In *International Symposium on Future Software Technology '99*, pages 119–124, October 1999.

[53] Akito Monden, Hajimu Iida, Ken ichi Matsumoto, Koji Torii, and Katsuro Inoue. A practical method for watermarking java programs. In *COMPSAC '00: 24th International Computer Software and Applications Conference*, pages 191–197, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0792-1.

[54] Ginger Myles. Using software watermarking to discourage piracy. *Crossroads - The ACM Student Magazine*, 2004. URL http://www.acm.org/crossroads/xrds10-3/watermarking.html. Accessed: 21 March, 2009.

[55] Ginger Myles and Christian Collberg. Software watermarking through register allocation: Implementation, analysis, and attacks. In *International Conference on Information Security and Cryptology*, volume 2971/2004 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003. ISBN 978-3-540-21376-5.

[56] Ginger Myles and Christian Collberg. Software watermarking via opaque predicates: Implementation, analysis, and attacks. In *ICECR-7*, 2004.

[57] Ginger Myles, Christian Collberg, Zachary Heidepriem, and Armand Navabi. The evaluation of two software watermarking algorithms. *Softw. Pract. Exper.*, 35(10):923938, 2005. ISSN 0038-0644.

[58] Jasvir Nagra, Clark Thomborson, and Christian Collberg. A functional taxonomy for software watermarking. In Michael J. Oudshoorn, editor, *Aust. Comput. Sci. Commun.*, pages 177–186, Melbourne, Australia, 2002. ACS.

[59] Gleb Naumovich and Nasir Memon. Preventing piracy, reverse engineering, and tampering. *Computer*, 36(7):6471, 2003. ISSN 0018-9162.

[60] Jens Palsberg and Di Ma. Javawiz, 2000. URL http://www.cs.purdue.edu/homes/madi/wm/. No longer available.

[61] Z. Pervez, Noor-ul-Qayyum, Y. Mahmood, and H.F. Ahmad. Semblance based disseminated software watermarking algorithm. In *Computer and Information Sciences, 2008. ISCIS '08. 23rd International Symposium on*, pages 1–4, 2008.

[62] Zhou Ping, Chen Xi, and Yang Xu-Guang. The software watermarking for tamper resistant radix dynamic graph coding. *Inform. Technol. J.*, 9:1236–1240, June 2010.

[63] Mila Dalla Preda, Matias Madou, Koen De Bosschere, and Roberto Giacobazzi. Opaque predicates detection

by abstract interpretation. In Michael Johnson and Varmo Vene, editors, *AMAST*, volume 4019 of *Lecture Notes in Computer Science*, pages 81–95. Springer, 2006. ISBN 3-540-35633-9.

[64] Gang Qu and Miodrag Potkonjak. Analysis of watermarking techniques for graph coloring problem. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 190–193, San Jose, California, United States, 1998. ACM. ISBN 1-58113-008-2.

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

[66] Gang Qu and Miodrag Potkonjak. Fingerprinting intellectual property using constraint-addition. In *Design Automation Conference*, pages 587–592, 2000.

[67] G. Ramalingam. The undecidability of aliasing. *ACM Trans. Program. Lang. Syst.*, 16(5):1467–1471, 1994.

[68] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978. ISSN 0001-0782.

[69] Peter R. Samson. Apparatus and method for serializing and validating copies of computer software, February 1994.

[70] Zonglu Sha, Hua Jiang, and Aicheng Xuan. Software watermarking algorithm by coefficients of equation. *Genetic and Evolutionary Computing, International Conference on*, 0:410–413, 2009.

[71] M. Shirali-Shahreza and S. Shirali-Shahreza. Software watermarking by equation reordering. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*, pages 1–4, 2008.

[72] Smardec. Allatori java obfuscator, September 2009. URL `http://www.allatori.com/`. Accessed: 2 April, 2010.

[73] Jose Sogiros. Is protection software needed watermarking versus software security. http://bb-articles.com/watermarking-versus-software-security, March 2010. URL `http://bb-articles.com/watermarking-versus-software-security`. Accessed: 13 April, 2010.

[74] Julien Stern, Gael Hachez, Francois Koeune, and Jean-Jacques Quisquater. Robust object watermarking: Application to code. In *Information Hiding Workshop '99*, pages 368–378, 1999.

[75] Smita Thaker. *Software Watermarking via Assembly Code Transformations*. Masters thesis, San Jose State University, 2004.

[76] Ramarathnam Venkatesan and Vijay Vazirani. Technique for producing through watermarking highly tamper-resistant executable code and resulting watermarked code so formed, May 2006. Microsoft Corporation, US Patent: 7051208.

[77] Ramarathnam Venkatesan, Vijay Vazirani, and Saurabh Sinha. A graph theoretic approach to software watermarking. In *Proceedings of the 4th International Workshop on Information Hiding*, 2001.

[78] Bill Venners. *Inside the Java Virtual Machine*. McGraw-Hill, Inc., New York, NY, USA, 1996. ISBN 0079132480.

[79] Wang Yong and Yang Yixian. A software watermark database scheme based on PPCT. In *CIHW2004*, 2004.

[80] S. Jamal H. Zaidi and Hongxia Wang. On the analysis of dynamic software watermarking. Technical report, 2009.

[81] J. Zhu, Y. Liu, and K. Yin. A novel planar IPPCT tree structure and characteristics analysis. *Journal of Software*, 5(3):344, 2010.

[82] Jianqi Zhu, Kexin Yin, and Yanheng Liu. A novel DGW scheme based on 2D_PPCT and permutation. *Multimedia Information Networking and Security, International Conference on*, 2:109–113, 2009.

[83] William Zhu and Clark Thomborson. Algorithms to watermark software through register allocation. In *Digital Rights Management. Technologies, Issues, Challenges and Systems*, volume 3919 of *Lecture notes in computer science*, pages 180–191, Berlin, Allemange, 2006. Springer. ISBN 978-3-540-35998-2.

[84] William Zhu and Clark Thomborson. Extraction in software watermarking. In Sviatoslav Voloshynovskiy, Jana Dittmann, and Jessica J. Fridrich, editors, *MM&Sec*, pages 175–181. ACM, 2006. ISBN 1-59593-493-6.

[85] William Zhu and Clark Thomborson. Recognition in software watermarking. In *Proceedings of the 4th ACM international workshop on Contents protection and security*, pages 29–36, Santa Barbara, California, USA, 2006. ACM. ISBN 1-59593-499-5.

[86] William Feng Zhu. *Concepts and Techniques in Software Watermarking and Obfuscation*. PhD thesis, The University of Auckland, 2007.