INAE Letters (2019) 4:65–75
https://doi.org/10.1007/s41403-018-0058-8

REVIEW ARTICLE

Software Watermarking: Progress and Challenges

Ayan Dey[1] · **Sukriti Bhattacharya**[2] · **Nabendu Chaki**[1]

Received: 25 April 2018 / Accepted: 25 September 2018 / Published online: 20 October 2018

Abstract
In this paper, we present a brief survey on software watermarking methods that explains the prospects and constraints of most software watermarking algorithms. We introduce a detailed classification of the existing software watermarking techniques, the associated attack models, along with a review of the current software watermarking tools. The reader can find a clear research path on software watermarking from 1996 to till date. We also critically analyzed the strength and weakness of the existing watermarking schemes and explored the research gaps where the future researchers can concentrate.

Keywords Software watermarking · Survey · Attack models

Introduction

Software codes are always under the threat of being stolen. Therefore, it requires the owner of the code to take extra action before releasing the software such as proof of ownership, one of the critical aspects of large-scale software development. Software watermarking was introduced to prove the ownership of the software. Watermarking techniques are used extensively on digital contents such as multimedia files. The concept is extended into the software industry and has seen a lot of research interest. In this paper, a detailed survey of the existing and newly proposed software watermarking techniques is presented. We classify the techniques based on different domains in which information is embedded or hidden.

Formally stated, the process of software watermarking is to embed a structure $\mathcal{W} \in \mathbb{W}$ into a program $\mathcal{P} \in \mathbb{P}$ such that, $\mathcal{W}$ can be reliably located and extracted from $\mathcal{P}$ even after $\mathcal{P}$ has been subjected to code transformations, optimization and even obfuscation. On the other hand, $\mathcal{W}$ is stealthy, and hard to tamper with. Embedding $\mathcal{W}$ into $\mathcal{P}$ does not adversely affect the performance of $\mathcal{P}$. At the same time, $\mathcal{W}$ has a mathematical property which allows us to argue that its presence in $\mathcal{P}$ is the result of deliberate actions. The software watermarking process can be formally defined as follows:

Definition 1 Given, a set of programs $\mathbb{P}$ and a set of watermarks (identifiers) $\mathbb{W}$, software watermarking consists of two functions; watermark embed, $\mathsf{WM}_{\mathsf{embed}} : \mathbb{P} \times \mathbb{W} \rightarrow \mathbb{P}$ and watermark detection, $\mathsf{WM}_{\mathsf{detect}} : \mathbb{P} \times \mathbb{W} \rightarrow \mathsf{BOOL}$, where $\mathsf{BOOL} = \{\mathsf{TRUE}, \mathsf{FALSE}\}$.

The following diagram (Fig. 1) shows how watermark embed and watermark detect work in practice:

Software watermarking techniques can be broadly classified as static and dynamic. The static techniques hide the watermark in the application executable itself. Besides, in dynamic watermarking technique, watermarks are stored in a programs execution state.

The paper is organized as follows: in Sect. 2, we classify the techniques based on different domains in which watermark is embedded. At the end of Sect. 2, we presented some statistics on the research trends. We analyzed the strength of individual techniques concerning a set of watermarking attacks in Sect. 3. In Sect. 4, we concluded the paper describing the limitations with research gaps in the belief that this survey opens a unique aspect of software watermarking research

✉ Ayan Dey
adakc_rs@caluniv.ac.in

Sukriti Bhattacharya
sukriti.bhattacharya@list.lu

Nabendu Chaki
nabendu@ieee.org

[1] University of Calcutta, Kolkata, India

[2] e-Science Unit, Luxembourg Institute of Science and Technology, Luxembourg, Luxembourg

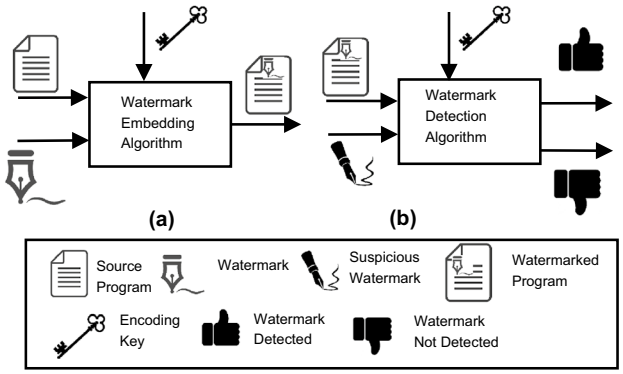

Fig. 1 Software Watermarking: **a** embed, **b** detect

Literature Survey

Software watermarking can be classified in different ways based on their approaches and types of publication (shown in Table 1). Before discussing some exciting works in details, we organize the ongoing research on software watermarking, in the last 23 years (1996–2018) in Table 2 based on the notations (Table 1).

Reordering-Based Approach

Software watermarking was introduced by Davidson and Myhrvold (1996). They proposed an algorithm that encodes watermark by reordering the basic blocks. Shirali-Shahreza and Shirali-Shahreza (2008) proposed a software watermark scheme based on the operations reordering. The operands used in an equation are altered based on the given information in a way such that the output of the equation remains the same. Sha et al. (2009) introduced a very similar scheme based on reordering the coefficients of operands. Regarding capability, Sha et al. (2009) is superior to Shirali-Shahreza and Shirali-Shahreza (2008). In 2011, Hamilton and Danicic (2011) present a code reordering-based survey on software watermarking. Later, (Sharma et al. 2012) proposed a static watermarking scheme for structural programming by reordering equations with function dependency-oriented sequencing (FDOS). Initially, they interchange safe operands of mathematical equations and then impose an ordering on the mutually independent functions with illogical dependency. Watermarking based on reordering is highly susceptible to semantic preserving transformation attacks.

Graph Theoretic-Based Approach

Several researchers proposed software watermark based on graph-theoretic approach. In 1998, Qu and Potkonjak (1998) introduced a scheme that hides the watermark by register allocation approach. The QP algorithm (named by Qu and Potkonjak (1998)) is a constraint-based watermarking algorithm that used graph coloring concept. Qu and Potkonjak (2000) presented a fingerprinting technique which is based on their earlier approach (Qu and Potkonjak 1998). QP algorithm adds an edge in a graph based on the watermark value and the resultant graph was used to capture the interaction between the program variables. They introduced (Qu and Potkonjak 1999) a one-way function and claim that meaningful message construction will be tough if a one-way encryption function is applied to the message. In 2004, Myles and Collberg (2004) proposed a new algorithm QPS, in SANDMARK (Collberg et al. 2003b) based on Qu and Potkonjak (1998). They select triplets of isolated vertices such that these will not affect the rest of the vertices in the graph. QPS has a prolonged data rate and is prone to various attacks like obfuscation-based attacks. In 2006, Zhu and Thomborson (2006b) described a modified version of QP algorithm, named QPI that requires actual inference graph with the original watermarked graph to extract the watermark. Later (Jiang et al. 2009) proposed a scheme combining QPI with public key encryption. They encrypted the watermark before embedding it.

In 1999, Collberg et al. proposed a dynamic watermarking that generates a graph during the program execution (Collberg and Thomborson 1999). Based on their approach, Krishnaswamy et al. (2000) proposed a scheme that embeds their watermark into the dynamic data structure. They also introduced a prototype named JavaWiz based on Collberg and Thomborson (1999), first practical tool for watermarking JAVA programs.

Hiding watermarks in various data structures became susceptible to subtractive attacks, since the attacker can eliminate the dummy data structures easily by semantics-preserving techniques. Kamel and Albluwi (2009) introduced a watermarking technique based on R-tree data structure and its variants. R-tree internally maintains

Table 1 Used Notations in our Paper

Sections	Full form (notations)
Approach	Path/branch (PB), reordering (RB), obfuscation (OB), chaos (CB), graph theoretic (GT), opaque predicate (OP), other methods (OT)
Publication type	Tool (T), Survey(SV), Research(R), Comparative study (CS)

Table 2 Software watermarking survey

Year	Reference	Techniques	Pub. type	Approach	Linked to
1996	Davidson and Myhrvold (1996)	S	R	RB	
1998	Qu and Potkonjak (1998)	S	R	RB, GT	
	Collberg and Thomborson (1998)	D	R	GT	
1999	Collberg and Thomborson (1999)	D	R	GT	
	Claudiu (1999)	S	R	GT	
	Pieprzyk (1999)	D	R	GT	
	Qu and Potkonjak (1999)	S	R	GT	
	Stern et al. (2000)	S	R	OB	
2000	Krishnaswamy et al. (2000)	D	T	GT	Collberg and Thomborson (1999)
	Monden et al. (2000)	S	R	OP	
	Cox et al. (2000)	–	SV	OT	
	Qu and Potkonjak (2000)	S	R	GT	
2001	Venkatesan et al. (2001)	S	R	GT	
	Collberg et al. (2004b)	D	T	GT	
2002	Collberg and Thomborson (2002)	S	R	OB	
	Arboit (2002)	S	R	OP	Monden et al. (2000), Collberg et al. (1998)
	Nagra et al. (2002)	–	SV	OT	
	Sion et al. (2002)	S	R	OT	
2003	Collberg et al. (2003a)	–	CS	GT	Collberg and Thomborson (1999), Venkatesan et al. (2001)
	Curran et al. (2003)	S	R	OT	
	Collberg et al. (2003b)	–	T	GT	
2004	Curran et al. (2004)	D	R	GT	
	Cousot and Cousot (2004)	S	R	OT	
	Myles and Collberg (2004)	S	R	GT	Qu and Potkonjak (1998)
	Tamada et al. (2004)	D	R	OT	
	Collberg et al. (2004a)	D	R	PB	
	Thomborson et al. (2004)	D	R	GT	
	Yong and Yixian (2004)	S	R	GT	
2005	Myles et al. (2005)	–	CS	OT	Davidson and Myhrvold (1996) , Monden et al. (2000)
	Zhu et al. (2005)	–	SV	OT	
	Myles and Jin (2005)	D	R	PB	
	Madou et al. (2005)	H	R	PB	Collberg et al. (2004a)
	Collberg and Sahoo (2005)	S	T	OB	Stern et al. (2000)
2006	Gil et al. (2006)	S	R	OT	
	Zhu and Thomborson (2006b)	S	R	GT	Qu and Potkonjak (1998)
	Myles and Collberg (2006)	S+D	R	OP	Arboit (2002)
	Gupta and Pieprzyk (2006)	D	T	PB	Myles and Jin (2005)
	Zhu and Thomborson (2006a)	S	R	OT	Collberg et al. (2004b)
	Liu et al. (2006)	S	R	CB	
2007	Collberg et al. (2007)	D	R	GT	
	Zhu (2007b)	–	SV	OB	
	Hopper et al. (2007)	–	SV	OT	
	Zhu (2007a)	–	CS	OT	Zhu and Thomborson (2006b), Zhu and Thomborson (2006a)
	Gupta and Pieprzyk (2007)	D	R	PB	Myles and Jin (2005), Gupta and Pieprzyk (2006)
2008	Mishra et al. (2008)	S	R	GT	
	Shirali-Shahreza and Shirali-Shahreza (2008)	S	R	RB	
	Dalla et al. (2008)	D	R	OT	
	Luo et al. (2008)	D	R	GT	Collberg and Thomborson (1999)

Table 2 (continued)

Year	Reference	Techniques	Pub. type	Approach	Linked to
2009	Kamel and Albluwi (2009)	D	R	GT	
	Zhu et al. (2009)	S	R	OT	
	Jianqi et al. (2009)	D	R	GT	Yong and Yixian (2004)
	Ke-xin et al. (2009)	D	R	CB	
	Collberg et al. (2009)	–	SV	GT	Venkatesan et al. (2001)
	Sha et al. (2009)	S	R	RB	
	Jiang et al. (2009)	D	R	GT	
2010	Bhattacharya and Cortesi (2010)	S	R	OT	
	Chroni and Nikolopoulos (2010)	S	R	GT	
	Zaidi and Wang (2010)	–	SV	OT	
	Zeng et al. (2010)	S	R	OB	
2011	Hamilton and Danicic (2011)	S	SV	RB	
	Chroni and Nikolopoulos (2011a)	S	R	GT	
	Chroni and Nikolopoulos (2011b)	S	R	GT	Collberg et al. (2003a)
	Zeng et al. (2011)	S	R	OB	Stern et al. (2000), Zeng et al. (2010)
2012	Chroni and Nikolopoulos (2012b)	S	R	GT	Chroni and Nikolopoulos (2010), Collberg et al. (2003a)
	Chroni and Nikolopoulos (2012a)	S	R	GT	
	Xu and Xiang (2012)	S	R	OT	
	Sharma et al. (2012)	S	R	RB	
	Yu et al. (2012)	S	R	OT	
2013	Fallis (2013)	D	R	OT	
	Chionis et al. (2013a)	D	R	OP, GT	
	Alitavoli et al. (2013)	S	R	CB	Arboit (2002), Monden et al. (2000)
	Chan et al. (2013)	D	R	GT	
2014	Bentoa et al. (2014)	S	R	GT	Collberg et al. (2003a), Chroni and Nikolopoulos (2012a)
	Chionis et al. (2014)	D	R	GT	Chroni and Nikolopoulos (2012b), Chionis et al. (2013a), Chionis et al. (2013b)
	Patel and Pattewar (2014)	D	R	GT	Chan et al. (2013)
	Balachandran et al. (2014)	S	R	OB	
	Bento et al. (2004)	D	R	GT	
2015	Ma et al. (2015)	D	R	OT	
	Cohen et al. (2015)	–	SV	OT	Hopper et al. (2007)
	Kumar et al. (2015)	–	CS	OT	
	Tian et al. (2015)	D	R,T	OT	
	Zong and Jia (2015)	S	R	OT	
2016	Mpanti and Nikolopoulos (2016)	S	R	GT	Chroni and Nikolopoulos (2010), Chroni and Nikolopoulos (2011a), Chroni and Nikolopoulos (2012a)
	Chen et al. (2016)	D	R	GT	
2017	Preda and Pasqua (2017)	–	R	OT	
	Chen et al. (2017a)	S	R	OT	
	Chen et al. (2017b)	D	R	PB	
	Nazir et al. (2017)	–	R	OT	
	Chen et al. (2017c)	D	R	PB	
2018	Wang et al. (2018)	D	R	OT	

the redundancy in the order of entries on the stored objects. Venkatesan et al. (2001) introduced a watermarking scheme based on the CFG modifications with some addition of specially marked node that stores the watermark values. This scheme is also suitable for solving the tamper resistance problem. In Collberg et al. (2009),

authors implemented the GTW (Graph-Theoretic Watermarking) algorithm proposed by Venkatesan et al. (2001). In 2008, a threshold-based scheme is proposed in Luo et al. (2008) that can cover up the weakness of Collberg and Thomborson (1999). Later, (Collberg et al. 2003a) focuses on two graph-based watermarking algorithms (Collberg and Thomborson 1999) and (Venkatesan et al. 2001). As per the study, Reducible Permutation Graph (RPG) and Planted Planar Cubic trees (PPCTs) offer the highest potential (Collberg et al. 2003a). PPCT is a binary tree where every internal node has a degree two (except the root). PPCT is used to represent a numeric value for constant encoding tamper proofing method (Thomborson et al. 2004). PPCT graphs have a lower bit rate and high resiliency than other graph-theoretic techniques. Collberg and Thomborson (1999), represented watermark by a number, encoded in the graph topology. This topology may adopt the structures like Radix, PPCT and improved PPCT. Krishnaswamy et al. (2000) explain the relationship between the watermark integer with the PPCT structure. But it fails to represent a long integer while used as a watermark. Therefore, an improved PPCT that merges the prospects of radix with PPCT is proposed in Yong and Yixian (2004).

He (2002) explained how Constant Encoding technique can be included in dynamic graph watermarking systems (He 2002). Mishra et al. (2008) proposed a method that watermarked the entire program for java class files that is almost similar to the scheme proposed by Venkatesan et al. (2001). The only difference is that (Venkatesan et al. 2001) marks the code segment; on the other hand, no marking is done in Mishra et al. (2008). In Chan et al. (2013), heap graph-based software birthmark techniques on JavaScript programs were used. They realize that lack of semantic-level integration between the program and watermark is the major problem faced by dynamic watermarking. Later, Chen et al. (2017b) introduced a control flow obfuscation to build a dynamic software watermarking framework. Watermark is divided into some fragments. Each fragment is stored in each obfuscated branch. Interestingly, control flow obfuscation connect obfuscated branches to a hidden execution path that is associated with internal program logic. In 2015, Tian et al. (2015) introduced a dynamic key instruction sequences-based technique for software birthmark. This instruction sequence can be extracted from the executable code and resilient from compiler optimization and obfuscation attacks. Another watermarking scheme (Chen et al. 2017a) uses the feature of Java reflection to find out the method names in a Java Code used to encode watermark bits.

In 1999, Pieprzyk (1999) used copyright fingerprinting to solve the problem of copyright protection for classified software identities. A dynamic fingerprinting scheme, proposed by Collberg et al. (2007), is embedded inside a graph during program execution. Comparing to static fingerprinting, dynamic fingerprint is more robust against obfuscation and optimization (Collberg et al. 2007).

In 2010, Chroni and Nikolopoulos (2010) proposed an algorithm for encoding watermark integers as self-inverting permutations (SIP). This algorithm takes a number and transforms it into SIP and from SIP to the number again within O(n) time, where n is the length of the binary representation of the number. The same authors also proposed a method for converting watermark integers into a graph through SIP (Chroni and Nikolopoulos 2011b). Based on this, Chroni and Nikolopoulos (2012b) presents an efficient scheme for encoding numbers as RPG (Chroni and Nikolopoulos 2010). Later, they extended (Chroni and Nikolopoulos 2012a) for embedding a watermark graph into an application program using its specific functions. Bentoa et al. (2014) formally characterizes the class of canonical RPG based on their encoding–decoding function, algorithm, etc. (Chroni and Nikolopoulos 2011a). Later, (Chionis et al. 2013b) presented a dynamic watermarking for embedding RPG into an application.

Obfuscation-Based Approach

In 2002, Collberg and Thomborson (2002) explained three different types of attacks on the intellectual properties of software. Defense against reverse engineering, piracy, and tampering is done by obfuscation watermarking and tamper proofing, respectively. Zhu (2007b) introduced a survey on software watermarking and obfuscating followed by a set of watermarking extraction and recognition algorithms. In 2000, SHKQ (named Stern–Hachez–Koeune–Quisquater) technique was proposed and applied on x86 assembly code (Stern et al. 2000). This algorithm is based on semantic preserving code transformations and code reordering for embedding watermark. Later, Collberg and Sahoo (2005) analyzes this by implementing it on SANDMARK framework (Collberg et al. 2003b) targeting Java bytecode. An obfuscation-based algorithm makes the control flow graph more complicated and leaves it harder for the reverse engineer to disassemble the code successfully. Zeng et al. (2010) proposed a robust scheme based on obfuscation that can resist various attacks. Later, they modified this in Zeng et al. (2011) and compared the improved proposal with Stern et al. (2000). Experimental results show that their schemes are susceptible to the high data rate and low embedding overhead.

Path- or Branch-Based Approach

Path-based watermarking introduced by Collberg et al. (2004a) embed the watermark in the runtime branch structure of the program. Such a scheme is less susceptible to

statistical attacks. Some watermarking techniques fail to address the problem of both static and dynamic attacks or hybrid (static–dynamic) attacks (Madou et al. 2005). Madou et al. discuss the potential of each attack through which an attacker can modify or control the program. They followed the path-based approach (Collberg et al. 2004a) as a case study.

Myles and Jin (2005) introduced branch-based software watermarking. They converted the unconditional jump statements by call instruction into a fingerprint branch function (FBF) that gives the target addresses of the jump instructions. It acts as an integrity checker as if the program tampers with an incorrect target address will be generated. Later, Gupta and Pieprzyk (2006) proposed a low-cost automated attack based on stack pointer modifications. This attack removes the fingerprint and integrity check, [described in Myles and Jin (2005)] generating code from the program after disassociating the target address. The modified scheme (Gupta and Pieprzyk 2007) is based on (Myles and Jin 2005) such that it can withstand with debugging attacks mentioned in Gupta and Pieprzyk (2006). This algorithm makes function manipulations more difficult than before.

Opaque Predicate-Based Approach

Opaque predicate-based software watermarking was initially proposed by Collberg et al. (1998). Later, Monden et al. (2000) proposed an approach that uses a dummy method as an authority mark into the Java source code. This technique is improved upon by Collberg et al. (1998), by protecting the never executed call to the method using opaque predicate. This approach easily avoids methods from being eliminated by the dead code. Later, Arboit (2002) proposed a watermarking method for java programs using an opaque predicate. Finally, Alitavoli et al. (2013) proposed a novel method for watermarking Java programs that are resilient to decompile-recompile and obfuscation based attacks. Consequently, Myles and Collberg (2006) evaluated two watermarking methods in SANDMARK (Collberg et al. 2003b). They conclude that the Arboit algorithm (Arboit 2002) is stronger than the Collberg's algorithm (Collberg et al. 2003b), based on some properties like resiliency and stealth. Besides, the authors explored if dynamic methods are inherently more resilient to attacks than static algorithms. Chionis et al. (2013a) used opaque predicates in specific control statements to control the flow of selected function calls of the program.

Chaos-Based Approach

Watermarking through chaos encryption is well defined by Ke-xin et al. (2009). They introduced a watermarking technique based on Shamir threshold and chaotic encryption. Thus, we require minimal information to extract the actual watermark. This method is resilient to various attacks except for the BLOAT (Bytecode-level Optimizer and Analysis Tool) (Nystrom 1998) and additive attacks. Similarly, Liu et al. (2006) proposed a watermarking scheme using an anti-reversing technique with the idea from Easter Egg. This chaotic system disperses watermark over the entire program to gain global protection for the program. Watermark is applied into the executable code; thus, recompilation is not required. Hence, the overall efficiency is improved.

Others Algorithms

Bhattacharya and Cortesi (2010) proposed a novel method for watermarking C source code, based on semantics through a hidden permutation first on local identifiers followed by the functions present in the source code. Since the scheme hides the watermark construction technique along with its location, it is resilient from the collusive attack. Similarly, Zero Watermarking embeds watermark information having no additional nodes (Xu and Xiang 2012). They hide the watermark information by code obfuscation. Cousot and Cousot (2004) describe an abstract interpretation-based fingerprint method, where static analysis of program semantics exposes the fingerprint. Watermarking based on various threads, locks and thread contention is proposed by Nagra and Thomborson (2004). Thaker (2004) proposed a scheme inspired by the working principle of metamorphic viruses. This produces functionally equivalent distinct watermarked programs that become more resilient to some attacks. Gil et al. (2006) develop a new linear time compression algorithm that compresses programs based on their syntactical structure. A table that is formed while compressing a program is considered as a fingerprint. They also showed the automatic fingerprint detection mechanism using the LZW algorithm. Due to the sensitivity on identifiers, LZW is unable to deal with program semantics. Consequently, Cox et al. (2000) describes the relationship between watermarking properties like robustness, cost, false positive rate, etc. with watermarking applications. They showed that a set of standards could not be applied to all watermarking methods. Several software watermarks have been classified based on the purpose and their properties (Nagra et al. 2002). Another useful survey (Zhu et al. 2005) describes the taxonomy, attack models and algorithms of software watermarking. In 2005, Myles et al. (2005) empirically evaluated two watermarking methods (Davidson and Myhrvold 1996) and (Monden et al. 2000), in SANDMARK (Collberg et al. 2003b) based on basic watermarking properties. Similarly, Sion et al. describe the fundamental watermarking issues on numeric data sets through resilient information hiding (Sion et al. 2002). Hamilton et al. present a categorical survey on static software watermarking approaches (Hamilton

and Danicic 2011). Zhu and Thomborson (2006a) describe the representative sets and degree which characterizes an extractable embedding algorithm. Later, they improved on his previous work (Zhu and Thomborson 2006a) to define the concepts of informed recognition corresponding to embedding algorithms (Zhu 2007a). Another watermarking framework for Java programs uses a signal detection model (Curran et al. 2003). Yu et al. (2012) proposed a scheme for software protection in the cloud platform. A dynamic software watermarking scheme based on exception handling (Wang et al. 2018) encodes binary watermark as an exception input sequence. It is embedded into the source code using the triggering condition and exception handling code.

Software Watermarking Tools

The following software watermarking tools as in Table 3 are currently dominating the watermarking research.

Critical Aspects of Software Watermarking

Resiliency

An attacker wants to get hold of the entire software or its logic, so that the software can be used for their purpose. Often several attempts are taken to weaken, remove or alter the watermark. Table 5 highlights some of the watermarking techniques that are compromised by the counterfeiting attacks. In this paper, we have considered 15 different types of attacks on 35 algorithms. Notations for each attack are described in Table 4. In Table 5, ✓ signifies that the corresponding algorithms is resilient to attacks and × signifies the non-resilience.

Table 4 Software watermarking attacks

Notation	Full form	Notation	Full form
A	Additive	OP	OPtimization
S	Subtractive	OB	OBfuscation
D	Distortive	RA	Re-allocation
ST	STatistical	DR	Decompile recompile
RO	Re-ordering	T	Tampering
RN	ReNaming	PM	Pattern matching
C	Collusive	RE	Reverse engineering

Limitations

In Sect. Resiliency, we presented some general kinds of attacks that reveal some significant limitations on the existing software watermarking schemes. However, new threats can compromise watermarking methods. In recognition attack, attacker creates confusion by attacking the detector module; cryptographic attack involves semi-reverse engineering way to hack the watermark. In protocol attack, attackers thwart a definitive identification of embedded watermark, by constructing false detector and false input.

Conclusion

Software watermarking is still one of the growing domains of research. Individually, the techniques proposed so far, in terms of security, often suffer from lack of robustness. What is missing yet is a rigorous theoretical framework. Earlier studies on software watermarking show that static techniques are highly susceptible to semantic preserving transformation attacks. Besides, such watermarks are prone to statistical attack. In contrast to this, we infer the following based on our study above,

Table 3 Software watermarking tools

Tools	Host language	Purposes
JavaWiz Collberg and Thomborson (1999)	JAVA codes[a]	Experimenting and testing several watermarking methods
Sandmark Collberg et al. (2003b)	JAVA BYTE codes[b]	Software watermarking, tamper proofing and code obfuscation
HYDAN El-Khalil and Keromytis (2004)	X86 Program[c]	Hides a message into an application by exploiting redundancy in X86 assembly
UWStego Collberg et al. (2004b)	X86 Program	User for watermarking java programs
Jmark Jdecode	JAVA class files[d]	Encoding and decoding a digital watermark into/from java class files
Jbirth	JAVA class files[e]	Extract birthmarks from java class files and compare them

[a] www.cs.purdue.edu/sss/projects/s3/

[b] www.cgi.cs.arizona.edu/~sandmark/

[c] www.crazyboy.com/hydan/

[d] www.se-naist.jp/jmark/

[e] www.se-naist.jp/jbirth/

Table 5 Resiliency of existing watermarking algorithms

Methods	Attacks														
	A	S	D	ST	RO	RN	DR	C	OP	OB	RA	DR	T	PM	RE
Davison 96	✓	×	×	✓	×			×			✓				
Collberg 98	✓	✓	✓		✓	✓									
Chiru 99			✓		✓		✓								
Collberg 99		×	✓						×	×					
Pieprzyk 99								✓							
Stern 99	✓	✓	✓		✓		✓			✓					
Cox 00	✓	✓	✓					✓							
Monden 00	✓	×	×				✓			✓					
Palsberg 00	×	×	×	✓											
Venkat 01			✓	×							×		✓		
Sion 02				✓							×	×			
Collberg 02													✓		✓
Arboit 02	×	×	✓												
Curran 04									×	×					
Cousot 04										×					
Tamada 04	✓	✓	✓							✓					✓
Nagra 04									✓	✓				✓	✓
Sahoo 05	×	×	✓					×				×			
Collberg 05			×	×					×	×					✓
Gupta 06		✓									✓				
Liu 06												✓			✓
Mishra 08													✓		
Jiang 09	×									✓					
Zeng 11														×	×
Xu 12			×											✓	✓
Jeon 12									✓	✓					
Chiru 13			×							×			×		
Chionis 13	✓	✓	✓												
De 14							✓			✓					
Chionis 14	✓	✓	✓				✓		✓	✓					
Ma 15	×		×							×					
Zong 15	✓	✓	✓												
Preda 17	✓	✓	✓					✓					✓		
Chen 17	✓	✓	✓	✓	✓										
Wang 18	✓	✓			✓										

1. Reordering-based algorithms are highly susceptible to semantic preserving transformation attacks.
2. Graph-Theoretic approaches are susceptible to obfuscation and program invariant attacks. Most of the data structure-based methods are prone to subtractive attacks. Algorithms based on numerical values tend to statistical attacks. Path-based techniques are not susceptible to statistical attack.
3. Obfuscation-based methods are resilient to reverse engineering attack. Tampering attacks are very hard for such techniques.
4. Opaque predicate-based methods are resilient to Decompile–Recompile and Obfuscation attacks.
5. Chaos-based approaches provide global protection from semantics preserving attacks.

Watermarking is a technique that is specifically designed for a variety of applications. Each application has a distinct trade-off between the basic properties of watermark like robustness, tamper resistance, etc. A specific set of standards could not be applied to all proposed methods. Thus, for a given set of standards, it is quite impossible to make a judgment among all methods. Similarly, it is quite impossible to resist all kinds of attacks. In this paper, we have tried to find out a relation between these attacks with a set of methods from the categorical view.

Acknowledgements This work was supported in part by the TCS Research Fellowship Award, granted to Ayan Dey and TEQIP Phase-III project of the University of Calcutta.

References

Alitavoli M, Joafshani M, Erfanian A (2013) A novel watermarking method for java programs. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing SAC '13. ACM, New York, NY, USA, pp 1013–1018

Arboit G (2002) A method for watermarking java programs via opaque predicates. ACM, Montreal, Canada, pp 102–110

Balachandran V, Keong NW, Emmanuel S (2014) Function level control flow obfuscation for software security. In: Eighth International Conference on Complex, Intelligent and Software Intensive Systems. IEEE, pp 133–140

Bento L, Boccardo D, Machado R, de Sá VGP, Szwarcfiter JL, Duque de Caxias R (2004) A randomized graph-based scheme for software watermarking. In: Information and Computater Systems Security. ACM, pp 30–41

Bentoa LM, Boccardob DR, Machadob RC, de Sáa VGP, Szwarcfitera JL (2014) Full characterization of a class of graphs tailored for software wateramrking. submitt J Comput Syst Sci

Bhattacharya S, Cortesi A (2010) Zero-knowledge software watermarking for c programs. In: Advances in communication, network, and computing, IEEE Computer Society, p 282–286

Chan PP, Hui LC, Yiu SM (2013) Heap graph based software theft detection. IEEE Transactions on Information Forensics and Security 8(1):101–110

Chen J, Dai S, Chen J (2016) An improved software watermarking scheme based on ppct encoding. In: Computational intelligence and design. IEEE, pp 341–344

Chen J, Li K, Wen W, Chen W, Yan C (2017a) Software watermarking for java program based on method name encoding. In: Advanced intelligent systems and informatics, Springer, pp 865–874

Chen Z, Jia C, Xu D (2017b) Hidden path: dynamic software watermarking based on control flow obfuscation. In: Computational science and engineering (CSE) and embedded and ubiquitous computing (EUC), vol. 2, IEEE, p. 443–450

Chen Z, Wang Z, Jia C (2017c) Semantic-integrated software watermarking with tamper-proofing. Multimed Tools Appl 77(9):11159–11178

Chionis I, Chroni M, Nikolopoulos SD (2013) A dynamic watermarking model for embedding reducible permutation graphs into software. In: Security and Cryptography (SECRYPT), IEEE, p 1–12

Chionis I, Chroni M, Nikolopoulos SD (2013) Evaluating the waterrpg software watermarking model on java application programs. In: Informatics, ACM, p 144–151

Chionis I, Chroni M, Nikolopoulos SD (2014) Waterrpg: a graph-based dynamic watermarking model for software protection. arXiv :1403.6658

Chroni M, Nikolopoulos SD (2010) Encoding watermark integers as self-inverting permutations. In: Computer systems and technologies, ACM, p 125–130

Chroni M, Nikolopoulos SD (2011) Efficient encoding of watermark numbers as reducible permutation graphs. arXiv: 1110.1194

Chroni M, Nikolopoulos SD (2011) Encoding watermark numbers as cographs using self-inverting permutations. In: Computer systems and technologies, ACM, p 142–148

Chroni M, Nikolopoulos SD (2012) An efficient graph codec system for software watermarking. In: Computer software and applications, IEEE, p 595–600

Chroni M, Nikolopoulos SD (2012) An embedding graph-based model for software watermarking. In: Intelligent information hiding and multimedia signal processing, IEEE, p 261–264

Claudiu C (1999) Static software watermarking pp. 2–4

Cohen A, Holmgren J, Vaikuntanathan V (2015) Publicly verifiable software watermarking. IACR Cryptology 2015:373

Collberg C, Carter E, Debray S, Huntwork A, Linn C, Stepp M (2004) Dynamic path-based software watermarking. Programming Language Design and Implementation 39:107–118

Collberg C, Huntwork A, Carter E, Townsend G, Stepp M (2009) More on graph theoretic software watermarks: implementation, analysis, and attacks. Inform Softw Technol 51(1):56–67

Collberg C, Jha S, Tomko D, Wang H (2004) Uwstego: a general architecture for software watermarking. Tech. rep

Collberg C, Kobourov S, Carter E, Thomborson C (2003) Error-correcting graphs for software watermarking. In: Graph theoretic concepts in computer science, Springer, p 156–167

Collberg C, Myles G, Huntwork A (2003) Sandmark-a tool for software protection research. IEEE Secur Priv 99(4):40–49

Collberg C, Sahoo TR (2005) Software watermarking in the frequency domain: implementation, analysis, and attacks. J Comput Secur 13(5):721–755

Collberg C, Thomborson C (1998) On the limits of software watermarking. CS Dept, Auckland Univ, Tech. rep

Collberg C, Thomborson C (1999) Software watermarking: models and dynamic embeddings. In: Principles of programming lang., ACM, New york, p 311–324

Collberg C, Thomborson C, Low D (1998) Manufacturing cheap, resilient, and stealthy opaque constructs. In: Principles of programming lang., ACM, p 184–196

Collberg CS, Thomborson C (2002) Watermarking, tamper-proffing, and obfuscation: tools for software protection. IEEE Trans Softw Eng 28:735–746

Collberg CS, Thomborson C, Townsend GM (2007) Dynamic graph-based software fingerprinting. ACM Trans Program Lang Syst (TOPLAS) 29(6):35

Cousot P, Cousot R (2004) An abstract interpretation-based framework for software watermarking. In: Principles of programming languages, ACM, p 173–185

Cox IJ, Miller ML, Bloom JA (2000) Watermarking applications and their properties. In: Information technology: coding and computing, IEEE, p 6–10

Curran D, Cinneide M.O, Hurley N, Silvestre G (2004) Dependency in software watermarking. In: Information and communication technologies: from theory to applications, IEEE, p 311–324

Curran D, Hurley N, Cinneide MO (2003) Securing java through software watermarking. In: Principles and practice of programming in Java, ACM, p 145–148

Dalla Preda M, Giacobazzi R, Visentini E (2008) Hiding software watermarks in loop structures. In: Static analysis, Springer, p 174–188

Davidson R, Myhrvold N (1996) Method and system for generating and auditing a signature for a computer program. US Patent 5,559,884

El-Khalil R, Keromytis AD (2004) Hydan: hiding information in program binaries. In: Lopez J, Qing S, Okamoto E (eds) Information and communications security. ICICS 2004. Lecture notes in computer science, vol 3269. Springer, Berlin, Heidelberg, pp 187–199

Fallis A (2013) Dynamic software watermarking by alering the numeric results of the program. J Chem Inform Modeling 53:1689–1699

Gil J, Gorovoy A, Itai A (2006) Software fingerprinting. In: Information technology: research and education, p 69–73

Gupta G, Pieprzyk J (2006) A low-cost attack on branch-based software watermarking schemes. In: Digital watermarking, Springer, p 282–293

Gupta G, Pieprzyk J (2007) Software watermarking resilient to debugging attacks. J Multimed 2(2):10–16

Hamilton J, Danicic S (2011) A survey of static software watermarking. In: World congress on internet security, IEEE, p 100–107
He Y (2002) Tamperproofing a software watermark by encoding constants. Master's thesis CS Dept, Auckland Univ
Hopper N, Molnar D, Wagner D (2007) From weak to strong watermarking. In: Theory of cryptography, Springer, p 362–382
Jiang Z, Zhong R, Zheng B (2009) A software watermarking method based on public-key cryptography and graph coloring. In: Genetic and evolutionary computing, Springer, p 433–437
Jianqi Z, YanHeng L, KeXin Y (2009) A novel dynamic graph software watermark scheme. In: Education Technology and Comp. Sci., vol. 3, pp. 775–780. IEEE
Kamel I, Albluwi Q (2009) A robust software watermarking for copyright protection. Comput Secur 28(6):395–409
Ke-xin Y, Ke Y, Jian-qi Z (2009) A robust dynamic software watermarking. In: Information technology and computer science, vol. 1, IEEE, p 15–18
Kumar K, Kehar V, Kaur P (2015) A comparative analysis of static java bytecode software watermarking algorithms. African J Comput ICT 8(4):201–208
Liu F, Lu B, Luo X (2006) A chaos-based robust software watermarking. In: Information security practice and experience, Springer, p 355–366
Luo YX, Cheng JH, Fang DY (2008) Dynamic graph watermark algorithm based on the threshold scheme. In: Information science and engineering, ISISE'08, vol. 2, IEEE, p 689–693
Ma H, Lu K, Ma X, Zhang H, Jia C, Gao D (2015) Software watermarking using return-oriented programming. In: Information, computer and communications security, ACM, p 369–380
Madou M, Anckaert B, De Sutter B, De Bosschere K (2005) Hybrid static-dynamic attacks against software protection mechanisms. In: Digital rights management, ACM, p 75–82
Mishra A, Kumar R, Chakrabarti P (2008) A method-based whole-program watermarking scheme for java class files. J Artic
Monden A, Iida H, Matsumoto K, Torii K, Inoue K (2000) A practical method for watermarking java programs. In: 24th international computer software and applications conference, pp 191–197, October 25–28, 2000
Mpanti A, Nikolopoulos SD (2016) Graph-structured watermarking using bitonic sequences of self-inverting permutations. In: Informatics, ACM, p 131–136
Myles G, Collberg C (2004) Software watermarking through register allocation: implementation, analysis, and attacks. In: Lim JI, Lee DH (eds) Information security and cryptology - ICISC 2003. ICISC 2003. Lecture notes in computer science, vol 2971. Springer, Berlin, Heidelberg, pp 274–293
Myles G, Collberg C (2006) Software watermarking via opaque predicates: implementation, analysis, and attacks. Elec Commerce Res 6(2):155–171
Myles G, Collberg C, Heidepriem Z, Navabi A (2005) The evaluation of two software watermarking algorithms. Softw Pract Exp 35(10):923–938
Myles G, Jin H (2005) Self-validating branch-based software watermarking. In: Info. hiding, Springer, p 342–356
Nagra J, Thomborson C (2004) Threading software watermarks. In: Info. hiding, Springer, p 208–223
Nagra J, Thomborson C, Collberg C (2002) A functional taxonomy for software watermarking. In: Australian computer science communications, vol. 24, Australian Computer Society, Inc, p 177–186
Nazir S, Shahzad S, Riza LS (2017) Birthmark-based software classification using rough sets. Arabian J Sci Eng 42(2):859–871
Nystrom NJ (1998) Bytecode level analysis and optimization of java classes. Master's thesis
Palsberg J, Krishnaswamy S, Kwon M, Ma D, Shao Q, Zhang Y (2000) Experience with software watermarking. In: Computer security applications. IEEE, pp 308–316
Patel SJ, Pattewar TM (2014) Software birthmark based theft detection of javascript programs using agglomerative clustering and frequent subgraph mining. In: Embedded systems (ICES), IEEE, p 63–68
Pieprzyk J (1999) Fingerprints for copyright software protection. In: Information security. ISW 1999. Lecture Notes in Computer Science, vol 1729. Springer, Berlin, Heidelberg, pp 178–190
Preda MD, Pasqua M (2017) Software watermarking: a semantics-based approach. Electron Notes Theoretical Comput Sci 331:71–85
Qu G, Potkonjak M (1998) Analysis of watermarking techniques for graph coloring problem. In: Computer-aided design, ACM, p 190–193
Qu G, Potkonjak M (1999) Hiding signatures in graph coloring solutions. In: Info. Hiding, vol. 1768, pp. 348–367. Springer
Qu G, Potkonjak M (2000) Fingerprinting intellectual property using constraint-addition. In: Annual design automation, ACM, p 587–592
Sha Z, Jiang H, Xuan A (2009) Software watermarking algorithm by coefficients of equation. In: Genetic and evolutionary computing, Springer, p 410–413
Sharma B, Agarwal R, Singh R (2012) An efficient software watermark by equation reordering and fdos. In: Soft computing for Prob. Sol., Springer, p 735–745
Shirali-Shahreza M, Shirali-Shahreza S (2008) Software watermarking by equation reordering. In: Information and communication technologies: from theory to applications, IEEE, p 1–4
Sion R, Atallah M, Prabhakar S (2002) On watermarking numeric sets. In: Digital watermarking, Springer, p 130–146
Stern JP, Hachez G, Koeune F, Quisquater JJ (2000) Robust object watermarking: application to code. In: Info. hiding, Springer, p 368–378
Tamada H, Okamoto K, Nakamura M, Monden A, Matsumoto Ki (2004) Dynamic software birthmarks to detect the theft of windows applications. In: Future software technology, vol. 20
Thaker S (2004) Software watermarking via assembly code transformations. Ph.D. thesis, S. J. S. University
Thomborson C, Nagra J, Somaraju R, He C (2004) Tamper-proofing software watermarks. In: Proceedings of the second workshop on Australasian information security, data mining and web intelligence, and software internationalisation, vol. 32. pp 27–36
Tian Z, Zheng Q, Liu T, Fan M, Zhuang E, Yang Z (2015) Software plagiarism detection with birthmarks based on dynamic key instruction sequences. IEEE Trans Softw Eng 41(12):1217–1235
Venkatesan R, Vazirani V, Sinha S (2001) A graph theoretic approach to software watermarking. In: Info. hiding, vol. 2137, Springer, p 157–168
Wang Y, Gong D, Lu B, Xiang F, Liu F (2018) Exception handling-based dynamic software watermarking. IEEE Access 6:8882–8889
Xu G, Xiang G (2012) A method of software watermarking. In: Systems and Informatics, IEEE, p 1791–1795
Yong W, Yixian Y (2004) A software watermark database scheme based on ppct. CIHW, 2004
Yu Z, Wang C, Thomborson C, Wang J, Lian S, Vasilakos AV (2012) A novel watermarking method for software protection in the cloud. Softw Prac Exp 42(4):409–430
Zaidi SJH, Wang H (2010) On the analysis of software watermarking. In: Software technology and engineering, IEEE, vol. 1, p 1–26
Zeng Y, Liu F, Luo X, Yang C (2010) Robust software watermarking scheme based on obfuscated interpretation. In: Multimedia

information networking and security (MINES), IEEE, p 671–675
Zeng Y, Liu F, Luo X, Yang C (2011) Software watermarking through obfuscated interpretation: implementation and analysis. J Multimed 6(4):329–340
Zhu J, Wei Q, Xiao J, Wang Y (2009) A fragile software watermarking algorithm for content authentication. In: Information, computing and telecommunication, IEEE, p 391–394
Zhu W (2007) Informed recognition in software watermarking. In: Intelligence and security informatics, Springer, p 257–261
Zhu W, Thomborson C (2006) Extraction in software watermarking. In: Multimedia and security, ACM, p 175–181
Zhu W, Thomborson C (2006) Recognition in software watermarking. In: Contents protection and security, MCPS '06, ACM, p 29–36
Zhu W, Thomborson C, Wang FY (2005) A survey of software watermarking. In: Intelligence and security informatics, Springer, p 454–458
Zhu WF (2007) Concepts and techniques in software watermarking and obfuscation. Ph.D. thesis, ResearchSpace@ Auckland
Zong N, Jia C (2015) Software watermarking using support vector machines. In: Computer software and applications, vol. 2, IEEE, p 533–542