International Journal For Research In Advanced Computer Science And Engineering
ISSN: 2208-2107

APPLICATION OF WATERMARKING TO SOFTWARE PIRACY

Ekene Frank Ozioko

Department of Computer and Information Science, Enugu State University of Science and Technology, Enugu . (ekene.ozioko@esut.edu.ng)

ABSTRACT

Within the software industry software piracy is a great concern. In this article we address this issue through a prevention technique called software watermarking. Depending on how a software watermark is applied it can be used to discourage piracy; as proof of authorship or purchase; or to track the source of the illegal redistribution. 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 particular we analyze an algorithm originally proposed by Genevie`ve Arboitin. A Method for Watermarking Java Programs via Opaque Predicates. This watermarking technique embeds the watermark by adding opaque predicates to the application.We have found that the Arboit technique does withstand some forms of attack and has a respectable data-rate. However, it is susceptible to a variety of distortive attacks.One unanswered question in the area of software watermarking is whether dynamic algorithms are inherently more resilient to attacks than static algorithms. We have implemented and empirically evaluated both static and dynamic versions within

INTRODUCTION

The global revenue loss due to software piracy was estimated to be more than $50 billion in 2009. 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.

Software watermarking is just one of many techniques that is currently being studied to prevent or discourage software piracy and copyright infringement. The idea is similar to media watermarking where a unique identifier is embedded in image, audio, or video data through the introduction of errors not detectable by human perception. Due to the nature of software it is not possible to strictly apply the ideas found in media watermarking. Instead embedding an identifier in a piece of software must be done in such a way that the original functionality is maintained.

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. 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. 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.

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.

Definition 1. (Software watermarking System). Given a program P, a watermark w, and a key k, a software watermarking system consists of two functions:

embed (P,w,k) →P

recognize (P,k) → w.

There are two general categories of watermarking algorithms,(1) static and (2)dynamic. A dynamic algorithm relies on information gathered from the execution of the application to embed and recognize the watermark. Static algorithms only examine the static code and data oftheapplication.Avarietyoftechniqueshavebeenproposedforsoftwarewatermarkingbut there are few publications describing the implementation and evaluation of these algorithms. There are far more static watermarking algorithms than dynamic due to the multitude of locations where information can be hidden in an executable. For example, in a Java classfile a static watermark can be embedded in the constant pool table, method table, etc.

Davidson and Myhrvold proposed a static watermarking algorithm which embeds the watermark by reordering the basic blocks of a control flow graph. Venkatesan et al built on this idea in an algorithm which embeds the watermark by extending a method's control flow graph through the insertion of a subgraph. Monden et al proposed a technique which embeds the watermark in a dummy method through a specially constructed instruction sequence. Stern et al also consider instruction sequences for embedding the watermark. Their technique modifies instruction frequencies to represent the watermark. Qu and Potkonjakmade use of the graph coloring problem to embed a watermark in the register allocation of an application. The first dynamic watermarking algorithm, CT, was proposed by Collberg et al. In this technique the watermark is embedded through a graph structure which is built on the heap at runtime. A second technique by Cousot makes use of abstract interpretation to embed a watermark in values assigned to integer local variables during execution. Collberg et al proposed a dynamic path-based technique which embeds the watermark in the dynamic branching behavior of the application by modifying the sequence of branches taken and not taken on the secret input sequence. A final dynamic technique by Nagra and Thomborsonrelies on multi-threading to embed the watermark. Of the early algorithms very little has been published on their implementation and evalu- ation. There are a few existing implementations of the CT algorithm, such as the one within the SANDMARK framework and that by Palsberg et al. A recent dissertation by Hachez provides an analysis of the Stern algorithm, as does Sahoo. The Qu and Potkonjak technique was evaluated by Myles and Collberg et al provides an evaluation of the Venkatesan technique. SANDMARK is a research tool for studying software protection techniques and in particular software watermarking, code obfuscation, and tamper-proofing of Java bytecode. One of the goals of the SANDMARK project is to implement and evaluate all known software watermarking algorithms. The system includes a variety of tools that permit the study of

watermarking algorithms with respect to such properties as resiliency and stealth. Through the implementation and evaluation of known software watermarking algorithms we will be able to gain an understanding of what makes a software watermarking technique strong.

Figure 1 shows a conceptual diagram of a simple static watermarking system.

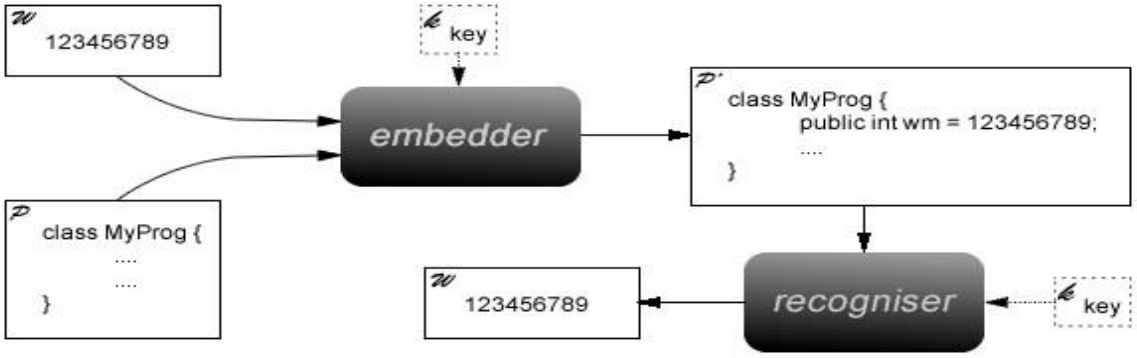

FIGURE 1

Opaque predicates

Opaque predicates were first presented by Collberg et al. [9] as a technique to aid in code ob- fuscation and later incorporated in a software watermarking technique proposed by Monden et al. [14, 15]. Informally, opaque predicates are inserted to make it difficult for an adversary to analyze the control-flow of the application. This makes it more difficult to identify that certain portions of the application are superfluous. For example, the Monden algorithm uses opaque predicates to disguise the fact that a dummy method is never invoked.

Table 1 Number theoretically true opaque predicates used in the implementation of the Arboit Algorithms

$\forall x, y \in \mathbb{Z}$	$7y^2 - 1 \neq x^2$
$\forall x \in \mathbb{Z}$	$2 \mid \lfloor \frac{x^2}{2} \rfloor$
$\forall x \in \mathbb{Z}$	$2 \mid x(x+1)$
$\forall x \in \mathbb{Z}$	$x^2 \geq 0$
$\forall x \in \mathbb{Z}$	$3 \mid x(x+1)(x+2)$
$\forall x \in \mathbb{Z}$	$7 \nmid x^2 + 1$
$\forall x \in \mathbb{Z}$	$81 \nmid x^2 + x + 7$
$\forall x \in \mathbb{Z}$	$19 \nmid 4x^2 + 4$
$\forall x \in \mathbb{Z}$	$4 \mid x^2(x+1)(x+1)$

Definition 2. (Opaque predicate). A predicate P is opaque at a program point p, if at point p the outcome of P is known at embedding time. If P always evaluates to True we write PT p, for False we write PF p , and if P sometimes evaluates to True and sometimes to False we write P? p.

Definition 3. (Opaque method). A boolean method M is opaque at an invocation point p, if at point p the return value of M is known at embedding time. If M always returns the value of True we write MT p , for False we write MF p , and if M sometimes returns True and sometimes False we write M? p.

Thekeychallengetousingopaquepredicatesoropaquemethodsistodesigntheminsuch a way that they are resilient to various forms of analysis. If an adversary can easily decipher thevalueofanopaquepredicateitprovidesverylittleprotectionforthesoftware.Avarietyof techniquessuchasusingnumbertheoreticresults,pointeraliases,andconcurrencyhavebee n suggested for the construction of opaque predicates . In addition to the number theoretic results, Arboit also suggests a technique for constructing a family of opaque predicates through the use of quadratic residues. Our current implementation of the Arboit algorithms uses number theoretically true opaque predicates and opaque methods. The nine we have implemented thus far can be seen in Table 1. An important aspect of the Arboit algorithms is that the opaque predicate library must remain secret. If an adversary knows even a few of the opaque predicates used in the embedding he may be able to identify them in the application and then remove them. None of the nine opaque predicates used in the current implementation are considered cryptographically secure or even resilient to analysis. While this does weaken the implemen- tation it does not invalidate the analysis in Section 5. The disadvantage of using these opaque predicates is that the algorithm is not as stealthy and is susceptible to manual attacks that will be elaborated on. As more sophisticated opaque predicates become available within the SANDMARK framework they will be used to embed the watermark in place of the simple ones in table 1

Arboit algorithm

Arboit proposed two watermarking techniques both based on opaque predicates. The first algorithm (henceforth GA1) is the basic insertion algorithm which directly uses the opaque predicates. To embed a watermark, w is split into k pieces, w0,...,w k−1,

and k branchingpoints, b0,...bk−1, are randomly selected throughout the application. At each branching point bi, either∧PTbi ,∨¬PT bi , or∨PF bi is appended to the predicate at that location. The bits ofthewatermarkareembeddedthroughtheopaquepredicatethathasbeenchosen.Withinthe opaque predicate the bits can be encoded either as constants or by assigning a rank to each of the opaque predicates. To recognize the watermark the application is scanned, extracting all identifiable opaque predicates. The bits of the watermark are then decoded from the opaque predicate. As an example, suppose our watermark is encoded in the opaque predicate x2 ≥ 0. A watermark could be embedded as follows:

```
class c{                                  class c{
   void ml(int a, int b){                    void ml(int a, int b){
      ...                                       ...
      if (a <= b) {...}          W              int x = 1;
      else{...}                                 if (a <= b)  &&
      ...                                           (x*x>= 0)){...}
   }                                            else{...}
}                                               ...
                                             }
                                          }
```

The second Arboit algorithm (henceforth GA2) is similar to GA1 except opaque methods are used to embed the watermark. Again k branching points b0,...,bk−1 are randomly selected throughout the application. For each bi, MT bi or MF bi is created and a method call is appended. The bits of the watermark are encoded in the opaque method through the opaque predicate that it evaluates. To recognize the watermark the application is scanned, extracting all opaque methods which are first identified through their signatures. Once a possible candidate has been identified the method body is examined to find the opaque predicate. To illustrate, suppose we use the same opaque predicate as above. Using GA2 the application would be transformed in the following way:

```
class c {                                 class c {
   void ml(int a, int b){                    boolean m2(){
      ...                                       int x = 1;
      if (a <= b) {...}          W              return (x*x >=0);
      else{...}                                 }
      ...                                       void ml(int a, int b){
   }                                               ...
}                                                  if ((a <= b) &&
                                                       m2()){...}
                                                   else{...}
                                                   ...
                                                }
                                          }
```

Arboit claims that GA2 is more secure. The main argument is that changing the signature ofamethodisdifficult.However,thisclaimisuntrueandSANDMARK includescodeobfusca-tions which can do just that.wewillshowthatGA1isinfactastrongeralgorithm than GA2. This claim demonstrates the importance of implementation and evaluation in the proposal of a software watermarking algorithm.

Implementation details

Our implementations of GA1 and GA2 follow from the algorithms presented by Arboit.A few modifications described below were made in an attempt to make the algorithms more resilient to attack. In addition, we developed and implemented dynamic versions of the algorithms.

Watermark encoding

Arboit proposed an encoding technique in which each piece of the watermark also includes an index value. By including the index value the watermark pieces can be recovered in any order. Our implementation also splits the watermark so that it can be recovered in any order, but the index value is not required. Prior to embedding the watermark w it is encoded as an integer and split into k pieces {w1,w2,...,w k} such that 0 ≤ wi ≤ n. The technique used to split the watermark relies on a 1-1 correspondence between a multiset S of size m (where S ={si :0≤ si ≤ n}) and combinations of size n chosen from m +n elements. Given this correspondence, the splitter enumerates combinations of n chosen from the m +n elements for some fixed n. By using this particular splitting technique the order of the pieces is unimportant. The k pieces of the watermark are encoded in the opaque predicates in one of two ways: through the use of constants in the predicate or by assigning a rank to each of the opaque predicates in the library. If the opaque predicate is a number-theoretic result, wi can be encoded:

1. in the constants contained in the predicate, or

2. by inserting new constants in the predicate.

For example, consider encoding the value 42 using the opaquely true predicate 4|x2(x + 1)(x +1).

Thispredicatehasaconstantvalueof6becauseitcontainstheconstants 4, 1, and 1. Thus the value 36 still needs to be encoded. This is accomplished by multiplying bothsidesby18whichproducestheopaquepredicate[(18)(4)]|[(18)x2(x +1)(x +1)].This technique does not change the value of the opaque predicate and it permits the encoding of any n ∈

N

. To encode an odd valued watermark select an opaque predicate that already has an odd constant value such as 2|x(x +1). Either technique for encoding the watermark using constants is valid, but using only the constants that are contained in the predicate is restrictive. For example, using the 9 opaque predicates in Table 1, only

the values{0,3,4,6,8,27,88}can be encoded. The disadvantage of inserting new constants is that it makes the opaque predicate more obvious. To encode wi using rank, each of the opaque predicates are assigned a value starting at 0. Using SANDMARK's library the values {0,...,8} can be encoded. While this technique is simple, it does require that the opaque predicate library be a fair size in order to be useful.

Watermark embedding

Theembeddingprocessisdependentonidentifyingasetbofpossiblebranchingpoints.This set is identified through preprocessing each method in the application. For each wi∈ w an opaque predicate PT bj or a call to an opaque method MT bj is appended to a selected bj∈ B. In an attempt to increase the strength of the algorithm we identify local variables in the method which can be used in the opaque predicate. These variables are identified through the use of a forward slice centered aroundbj.

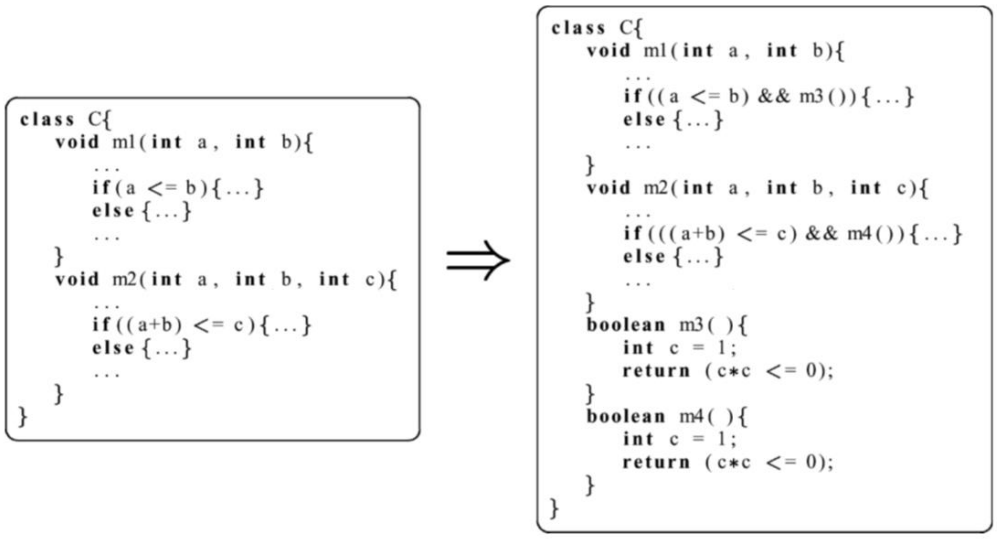

Fig. 1 Transformation without method reuse

The most significant advantage to using live variables in the opaque predicate (as opposed to inserting new variables) is that it aids in disguising the superfluous nature of the predicate. The current disadvantage to this technique is that it is not always possible to identify local variablescontainingintegersaroundaselectedbj.Thus,somebranchingpointsareunusable. This disadvantage will be alleviated as other types of opaque predicates become available. We were also able to add one more detail to the implementation that not only increases the stealth but decreases the overhead. To embed a watermark using GA2, k new methods are added to the application. This increase in code size could be unacceptable to size sensitive applications such as those on mobile devices. One

solution is to encode wi using rank and reuse the new methods that are added to the application. For example, without method reuse the example class C could be transformed into the class in Figure 1. With method reuse it is transformed into the class in Figure 2. This detail increases the stealth by further disguising the superfluous nature of the opaque method. Arboit discusses a technique to inhibit the adversary's ability to destroy the watermark using method overloading. If the adversary attempts to modify the types of the overloaded

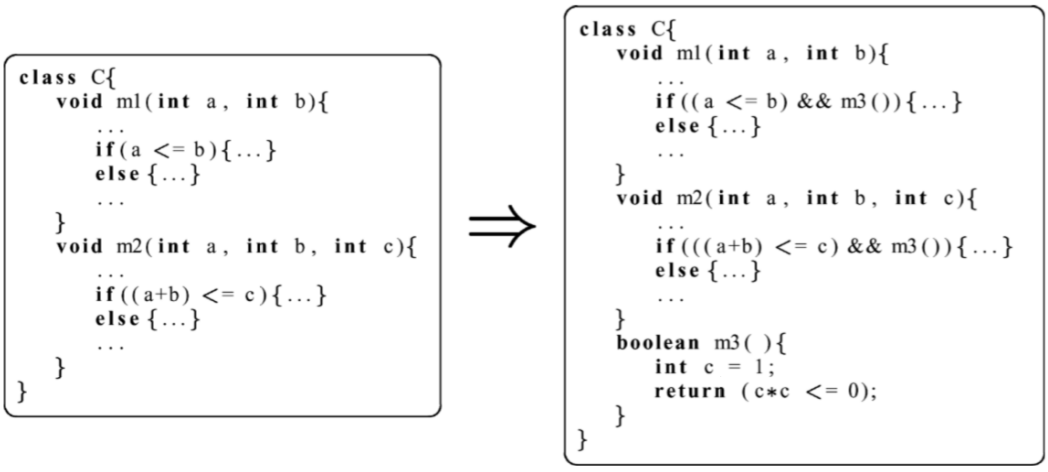

Fig. 2 Transformation with method reuse

method, overriding occurs which could lead to faulty behavior. The current implementation does not support this technique, but we will see that such a technique does not prevent watermark distortion in those instances where GA1 outperforms GA2.

Watermark recognition

The recognition procedure varies slightly depending on which embedding technique is used. Watermark recovery using GA1 involves an exhaustive search of each method. To identify sets of instructions that may be opaque predicates the basic blocks of the control flow graph (CFG) and expression trees are constructed. Each opaque predicate will end with an if instruction which can be found as the last instruction of a basic block. The instructions that comprise the expression tree for that if instruction are compared to the entries in the opaque predicate library. If the watermark was embedded using GA2 then each method is scanned looking for invoke instructions which call a method that has the same signature as one of the opaque methods. Currently all opaque methods have a return type of boolean and either 1 or 2 parameters of type int. In the case when opaque methods are not reused the

recognition process could have been simplified to checking the signature of each method. Unfortunately thisdoesnotyieldthecorrectnumberofpieceswhenmethods arereused.Withineachopaque method is an opaque predicate that is identified using the same technique as in GA1. If wi is encoded using rank, the rank of that particular opaque predicate is identified. If constants are used, the sum of the constants is extracted from the predicate. Once all possible wi have been identified the values are combined to produce the watermark value.

Dynamic arboit algorithms

Oneoftheyetunansweredquestionsintheareaofsoftwarewatermarkingiswhetherdynamic algorithms are inherently more resilient to attack than static algorithms. One technique to investigate this idea is to develop, implement, and evaluate a dynamic version of an already known static algorithm. To this end we have developed and implemented dynamic versions of GA1 and GA2 (DGA1 and DGA2 respectively). Dynamicalgorithmsmakeuseofaprogram'sexecutionstatetobothembedandrecognize a watermark. There are three different dynamic techniques: Easter Egg Watermarks, Data StructureWatermarks,andExecutionTraceWatermarks.DGA1andDGA2areexecution tracewatermarkingalgorithmsbecausethewatermarkisembeddedinthetraceoftheprogra m as it is run with a specific input. This input represents the user's secret key. For example, suppose the application is a Tic-Tac-Toe game. The order in which the X's and O's are placed on the game board becomes the secret key. The novel aspect of DGA1 and DGA2 is that the execution trace is used to identify the set of program branching points B instead of using randomly selected points. The motivating factor in this design is that the program will execute the original set of branching points when run with the secret key no matter how distorted an attacker makes the application. This assumption is based on the idea that most transformations that cause the execution to skip the branch will most likely alter the functionality of the application. Thus the dynamic nature will improve the algorithm's ability to withstand distortive attacks. The set B of program branching points is required for both the embedding and recognition phases. B is compiled by annotating the application prior to execution. The annotation phase is fully automated and consists of adding a special function call immediately before each

Evaluation

In order for a software watermarking technique to be effective against software piracy and copyright infringement it should be resilient against determined attempts at discovery and removal. Very little work has been done on evaluating the strength of software watermarking systems and thus a formal set of properties has yet to be established. Through our study of software watermarking algorithms using the SANDMARK system we have compiled the following properties which we believe aid in evaluating the strength of an algorithm [8, 13, 20]:

credibility: The recognition process should report a watermark that was embedded and should not report false watermarks.

data-rate: The algorithm should have a high data-rate to permit the embedding of a reason- ably sized secret message.

overhead: Embedding a watermark should have little impact on the performance of the application and the embedding/recognition procedure should not be costly. part protection: In order to protect the watermark it should be distributed throughout the application. resiliency: The watermark must be resilient against determined attempts at discovery and removal. In particular it should be resilient to three important types of attacks:

In a subtractive attack the attacker attempts to remove the watermark from the disassem- bled or de-compiled code. Through a manual or automated inspection of the code the attacker may be able to identify and remove a watermark with low transparency without damaging the application.

In an additive attack the attacker adds a new watermark to the already watermarked program in an attempt to cast doubt on which watermark was embedded first.

In a distortive attack a series of semantics-preserving transformations are applied to the software in an attempt to render the watermark unrecoverable but maintain the software's functionality and performance.

stealth: The embedded watermark should be difficult to detect; i.e. it should exhibit the same properties as the code or data around it.

We have evaluated both the static and dynamic versions of theArboit algorithm within SANDMARK with respect to each of the above properties. SANDMARK includes a variety of tools that an adversary may use to discover and/or remove a watermark. These tools include:

An obfuscation tool that permits the evaluation of resiliency of the watermark under distortive attacks.

Additional watermarking algorithms for studying additive attacks (and in the future for comparison purposes).

A bytecode viewer to display the watermarked bytecode and for manually examining the stealth of the watermark.

A statistics module that provides static statistics about an application, such as the number of methods, number of conditional statements, etc., which also aids in the evaluation of stealth.

ToevaluatethestaticGA1andGA2asetof11applicationsareusedwhichvaryinbothsize and complexity. Two of these 11 applications are also used for the dynamic algorithms: TTT (which is a Tic-Tac-Toe game) and JKeyboard (which allows a user to type using different alphabets). The evaluation of the dynamic algorithms requires applications that make use of user input. This is required so that different execution traces can be obtained. Details of 10 of the applications can be seen in Table 2. The 11th application is specjvm.

Credibility

The credibility of a watermarking algorithm is based on the accuracy of watermark recovery. An algorithm can have poor credibility if it recovers a watermark which was not embedded in the application (a false positive) or not recovering a watermark that was embedded (a false negative). To evaluate the algorithms with respect to this property we ran the recognition algorithms on non-watermarked and obfuscated versions of the benchmark applications. No false negatives or false positives were detected in any of the test cases.

Part protection

Theideabehindthepartprotectionpropertyistosplitthewatermarkintopiecesandspreadit acrosstheapplication.Thesplitwatermarkhasabetterchanceatsurvivalsinceitrequiresthat the attack target multiple locations in the application. Both the static and dynamic algorithms incorporate part protection by splitting w into k pieces and randomly distributing those pieces. It was previously mentioned that reusing the opaque methods provided an advantage by decreasing the overhead and increasing the stealth. Unfortunately this technique also decreases the part protection. If the opaque method was used to encode three of the 10 pieces of w removing the method has a higher impact than if only one piece was destroyed.

Resilience

There are three types of attacks that an adversary could launch in an attempt to destroy a watermark: subtractive, additive, and distortive.

Subtractive attacks

One of the first things that an adversary may do in an attempt to eliminate a watermark is decompile the application. Once the code has been decompiled the attacker can search for aspects of the code that look suspicious such as dummy methods. If the attacker is familiar with simple number theory properties he may realize that the watermark application contains opaque predicates. If they are removed the application will still function normally and the attacker has subverted the protection. This watermarking technique will always be susceptible to subtractive attacks but using stronger opaque predicates, such as ones that are notcommonlyknown,willmakeitharderfortheattackertodetectthewatermarkedsections.
In addition, maintaining the secrecy of the opaque predicate library will also improve the resiliency against subtractive attacks.

Table 5 Results from applying other watermarking algorithms to the applications watermarked using GA1, GA2, DGA1, and DGA2. We found that for all test cases the results were the same. A '+' indicates that the original watermark was recovered. A '−' indicates that the original watermark was destroyed

Watermarker	Embedded using GA1	Embedded using GA2
AddMethodField	+	+
GA1	−	−
GA2	−	−
QP	+	+
BogusExpressions	+	+
BogusSwitch	+	+
BogusInitializer	+	+
ConstantString	+	+
HatTrick	+	+
MethodRenamer	+	+
MondenWmark	+	+

Additive attacks

Additive attacks are used by an adversary when he is either unable to locate the watermarked code or unable to remove the watermarked code. This type of attack is used to cast doubt on the validity of the original watermark or to destroy the original all together. Table 5 shows the results from applying other watermarking algorithms in the SANDMARK system to the test cases that had been watermarked using GA1, GA2, DGA1, DGA2. We found that the original watermark is quite resistant to the application of an additional watermark. However, embedding a watermark using the same algorithm or one of the other GA's destroyed the original watermark. This occurred because the recognition procedure detected additional opaque predicates. In addition we discovered that both watermarks are unrecoverable if we apply GA1 then GA1, GA2 then GA2, or GA2 then GA1. Even though the original was destroyed, the attacker will not be able to embed his own watermark using one of these techniques. The same results occur with DGA1 and DGA2 except that applying DGA2 then DGA1 does not destroy both watermarks.

Distortive attacks

Distortive attacks are any semantics preserving code transformation, such as code obfus- cation or optimization algorithms. This type of attack is used to distort a watermark such that it is unrecoverable. The advantage of this attack over subtractive attacks is that the adversary need not know the exact location of the watermark. Rather, he can apply the transformation indiscriminately over the application. Through the application of the code obfuscations found in SANDMARK we discovered that GA1 is more resilient than GA2. This discovery contradicts the claim made in [5]. The author claims that GA2 is stronger since it is difficult to alter the signature of a method. The obfuscations Method 2R Madness, Primitive Promoter,andPromoteLocalsallmodifythesignaturesofthemethods in the application. It is possible that implementing the overloading technique described would improve the resiliency against Primitive Promoter and PromoteLocals.

Summary

Software piracy is an ongoing problem in the software industry. While there are some legal means to handle the problem they donot always target the guilty party.Software water markingis an additional technique that can be used in the battle. The technique makes proofof authorship or purchase possible and in some cases the source of the illegaldistribution can be identified. In this paper we provided an implementation andevaluation of two techniques proposed. In addition, we presented a novel extension of the technique to study static versus dynamic water marking algorithms.Through our analysis we showed that both GA algorithms can be defeated. We also showed that GA1 is a stronger algorithm than GA2. We basedthese conclusion on six properties.OftheseGA1hadaloweroverhead,wasmoreresilienttoattack, and demonstrated a higher degree of stealth. With respect to the remaining three properties the algorithms were equal. We also showed that the dynamic algorithms are only minimally stronger than the static versions. From this we conclude that it is not clear that converting a known static algorithm will improve the strength. However, this does not indicate that the class of dynamic algorithms is not inherently stronger.

References

[1] Business software alliance, http://www.bsa.org.

[2] Sandmark. http://www.cs.arizona.edu/sandmark/.

[3] Specjvm98 v1.04. http://www.specbench.org/osg/jvm98/.

[4] Aho, A. V., Sethi, R., & Ullman, J. D. (1988). Compilers: Principles, Techniques, and Tools. Addison- Wesley.

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