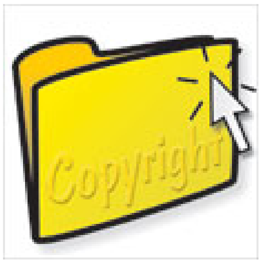
Crossroads
The ACM Student Magazine



Using Software Watermarking to Discourage Piracy

by ***Ginger Myles***

Introduction

Software piracy and copyright infringement are rapidly growing. Historically, the spread of pirated software required the transfer of a physical copy (i.e. a disk), limiting the rate of illegal software distribution. However, recent increases in network transfer rates and ease of access have eliminated the need for physical media based piracy. To compound the problem, software is being legally distributed in platform independent formats, such as Java bytecode and Microsoft's Intermediate Language (MSIL). These formats closely resemble source code, which can easily be reverse engineered and manipulated. Thus it is much easier for software pirates to bypass license checks. In addition, unscrupulous programmers can steal algorithmic secrets, which decreases their own production time and allows them to gain an edge on the competition.

There are legal ramifications associated with software piracy, such as statutory damages of up to $150,000 for each program copied [1]. However, these fines are often targeted at an unsuspecting end user and not at the person responsible for the piracy. When a person unknowingly purchases and uses an illegal piece of software it is often difficult to trace this software back to the guilty party. In addition, it is also hard to detect and prove that a dishonest programmer has taken advantage of a trade secret. The focus of this article is on software watermarking, a software-based technique developed to aid in piracy prevention and identification of the guilty party.

Piracy Prevention Techniques

Organizations such as the Business Software Alliance (BSA) [1] perform audits to verify that corporations are not using illegal software. Unfortunately, auditing does not identify an unknown software pirate or unethical programmer. In an attempt to curb software piracy, a variety of hardware and software techniques have been proposed. The hardware based approaches typically provide a higher level of protection; however, they are more cumbersome for the user and more expensive for the software vendor. Two such examples are tamper-proof hardware and dongles. Software-based solutions, such as code obfuscation, software tamper-proofing, and software watermarking, are cheaper but provide a lower level of protection.

Tamper-proof hardware aids in piracy prevention by providing a secure context and/or secure data storage. By executing the software in a secure environment the pirate is unable to gain access to the software. This technique prevents the attacker from observing the behavior of the software. The obvious drawback to this technique is the additional cost of requiring all users to have tamper-proof hardware. The second hardware based technique is a dongle which is a device distributed with the software. Possession of the the device proves ownership of the software. A dongle typically connects to an I/O port and computes the output of a secret function. Periodically the software queries the dongle. If the result of the query is the wrong output, the software reacts appropriately. There are two drawbacks to the use of dongles: (1) cost (a single dongle can cost at least $10) and (2) distribution of a dongle with software over the Internet is impractical.

Code obfuscation, a software-based solution, is a technique which aids in the prevention of reverse engineering through transformations that make the application more difficult to understand while preserving the original functionality. The idea is to obscure the readability and understandability of the program to such a degree that it is more costly for the attacker to reverse engineer the program than to simply recreate it. This particular technique was the focus of the Crossroads article "Protecting Java Code via Code Obfuscation" by Douglas Low [6]. The second software-based technique is software tamper-proofing. In this technique methods are employed to prevent the alteration of the program. For example, many programs contain license checks that prevent the user from using the software after a specific date. To prevent an attacker from removing the license check, tamper-proofing techniques are used that prevent the alteration. If an attacker does remove the license check then the tamper-proofing technique causes the software to fail. A third software-based technique is software watermarking, which we will discuss in detail.

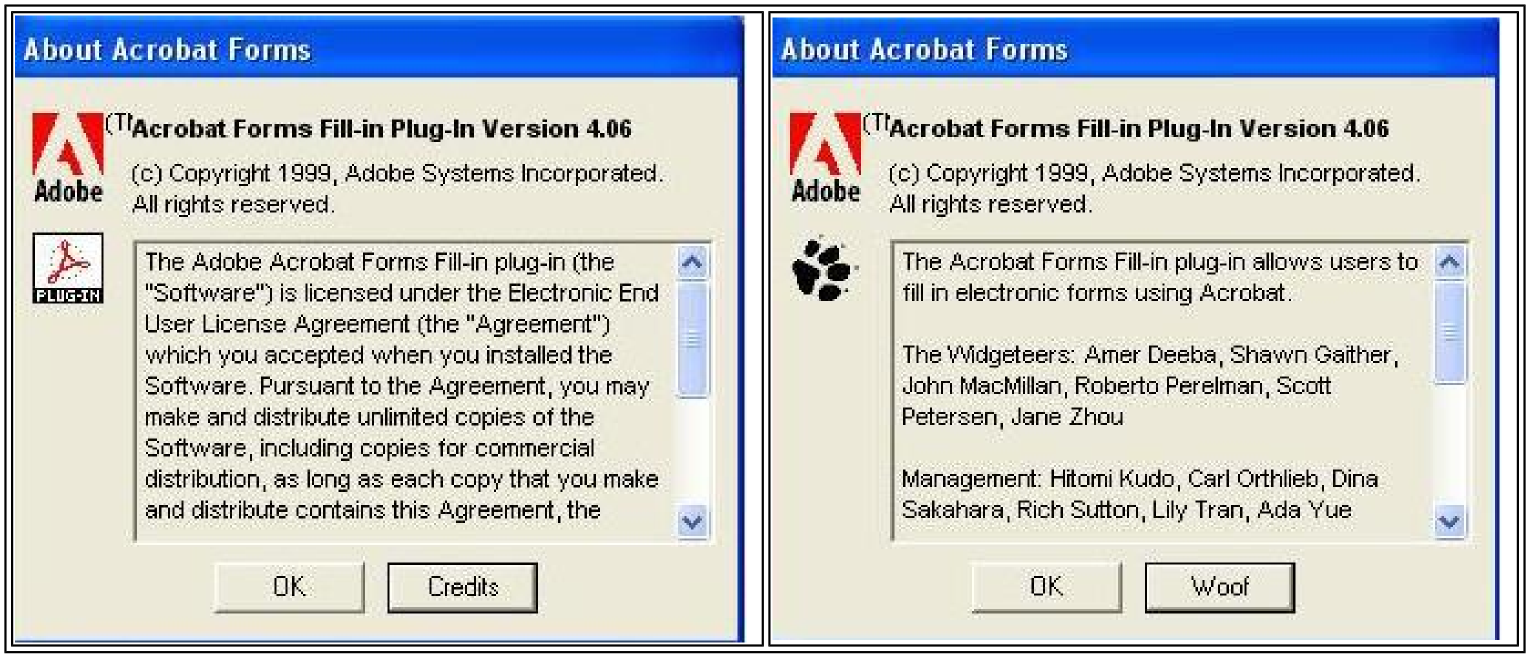
Software Watermarking

Software watermarking is a technique used to protect software from piracy. Unfortunately, watermarking alone does not prevent piracy. Instead it is used to discourage a user from illegally redistributing copies of the software. The general idea of software watermarking is very similar to media watermarking in which a unique identifier is embedded in images, audio, or videos through the introduction of errors which are undetectable by the human auditory system. Because the functionality of software is crucial to its success, the watermark must be embedded through techniques other than the introduction of errors.

Software watermarking embeds a unique identifier W (the "watermark") into a program P. If W uniquely establishes the author of P then W is considered a copyright notice. On the other hand, if W uniquely identifies the legal purchaser of P then W is a fingerprint. One of the most important aspects of a watermarking system is the use of a secret key. Through the use of the key the watermark is incorporated into the program producing a new program.

Currently, watermarking algorithms are described as either being static or dynamic. **Static watermarking algorithms** only make use of the features of an application that are available at compile-time, such as the instruction sequence or the constant pool table in a Java application, to embed a watermark. On the other hand, a **dynamic watermarking algorithm** relies on information gathered during the execution of the application to both embed and recognize the watermark. There are three different dynamic techniques: easter egg watermarks, data structure watermarks, and execution trace watermarks [3]. Some of the current research is focused on the study of static versus dynamic algorithms in order to determine if one is better than the other. As of now all that can be said is that the algorithms make use of different information to embed the watermark.

An easter egg watermark is a piece of code that can only be executed by a highly unusual input sequence. For example, in Adobe Acrobat Reader 4.0 select Help -> About Plug-ins -> Acrobat Forms and then hold Control+Alt+Shift while clicking on the credits button. This special input sequence will reveal the easter egg in **Figure** 1 that consists of three features: a dog bark, a credit button face that changes to say "woof", and an Adobe emblem that becomes a dog paw[12]. The drawback to using easter eggs to embed a watermark is that once they have been discovered, simple debugging techniques make it easy to locate and remove the watermark.



Acrobat Forms Fill-in Plug-In Version 4.06
(c) Copyright 1999, Adobe Systems Incorporated. All rights reserved.

The Adobe Acrobat Forms Fill-in plug-in (the "Software") is licensed under the Electronic End User License Agreement (the "Agreement") which you accepted when you installed the Software. Pursuant to the Agreement, you may make and distribute unlimited copies of the Software, including copies for commercial distribution, as long as each copy that you make and distribute contains this Agreement, the

Acrobat Forms Fill-in Plug-In Version 4.06
(c) Copyright 1999, Adobe Systems Incorporated. All rights reserved.

The Acrobat Forms Fill-in plug-in allows users to fill in electronic forms using Acrobat.

The Widgeteers: Amer Deeba, Shawn Gaither, John MacMillan, Roberto Perelman, Scott Petersen, Jane Zhou

Management: Hitomi Kudo, Carl Orthlieb, Dina Sakahara, Rich Sutton, Lily Tran, Ada Yue

Figure 1: Easter Egg Watermark found in Adobe Acrobat Reader 4.0

A data structure watermark embeds the watermark in the state of a program (such as the global, heap, and stack data) as the program is executed with a particular input. This technique is far more stealthy than an easter egg watermark since no output is produced. The third technique, execution trace watermarking, embeds the watermark within the trace of the application as it is executed with a particular input. This technique differs from the data structure watermark in that the watermark is embedded in the application's instructions or addresses instead of the application's state.

Examples

Embedding of a copyright notice or a fingerprint serve different purposes. The original creator of the software uses a copyright notice to prove ownership. This unique identifier is extremely helpful in proving that a program or a piece of a program is stolen. For example, suppose a programmer from Company B steals a secret module from Company A to decrease his own production time. If Company A can demonstrate that Company B's software contains their watermark then company A can prove that Company B is illegally profiting. Unfortunately, this type of watermark only proves that Company B was using the secret and not that they were the ones that stole it.

Tracking the source of illegal distribution requires a fingerprint, rather than a copyright notice, to link the particular copy to the original purchaser. To illustrate, suppose Alice sells Bob a copy of her software. Before she gives Bob the copy she embeds his credit card number. When Alice obtains a copy of her software, which she believes is pirated, she uses the recognize function with her secret key to extract the watermark. Since the watermark is Bob's credit card number, which uniquely identifies him, she can prove that Bob is the guilty pirate. **Figure 2** illustrates this scenario.

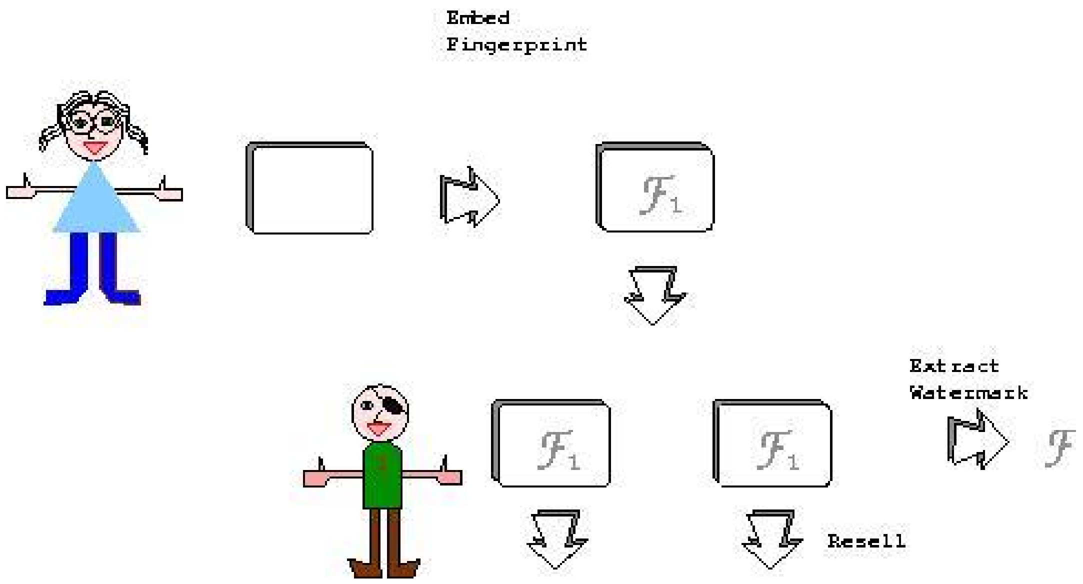


Figure 2: Alice embeds Bob's credit card number in order to identify Bob as the guilty pirate.

Companies create and distribute beta versions of their software. The distribution is targeted at a select handful of users and designed to solicit feedback. Generally, the release of the beta version to a user is contingent on the software remaining confidential. Although this agreement is made, a leak frequently occurs. The embedding of a unique fingerprint tied to the user permits the company to take legal action against the identified party.

Creators embed unique identifiers to assert ownership and/or trace the pirate after the fact. Software watermarking does not prevent piracy, but discourages the user from illegally copying the software by increasing the possibility that the pirate will get caught. One simple watermarking technique that illustrates the general idea is to embed the watermark as a field in the application. Suppose the string "wildcat" is embedded in `class C` below using this technique.

Before:
```
class C{
int a = 1;
void m1(){...}
void m2(){...}
}
```

After:
```
class C{
int a = 1;
int wildcat$ = 5;
void m1(){...}
void m2(){...}
}
```

In this particular example a special character, '$', was also added to aid in the recognition of the watermark. While this technique is simple to implement it is not effective in preventing piracy. Since many applications are distributed in formats that are easy to reverse engineer, an attacker could simply examine the source code, identify this as the watermark, and remove it. A second attack on this watermarking technique would be to use a simple code obfuscation that renames all of the identifiers in the application.

A variety of different software watermarking algorithms has been proposed in the past. One of the simplest techniques was proposed by Monden et al. [7]. This particular technique embeds the watermark in a dummy method that is added to the application. The embedding is accomplished through a specially constructed sequence of instructions. Because the inserted method is never executed there is flexibility in how the instructions are constructed which permits the embedding of any watermark. Stern et al. [11] also consider instruction sequences for embedding the watermark. Their technique modifies the frequency of the instructions through out the application to represent the watermark.

Instead of embedding the watermark at the instruction level, a watermark can be embedded by manipulating the control flow graph (CFG) of a method. Davidson and Myhrvold [4] proposed a technique which does just that. By rearranging the basic blocks of the CFG and then redirecting the control flow to maintain the correct functionality the watermark is embedded. Venkatesan et al. [13] also use a CFG to embed a watermark. In this algorithm a subgraph is inserted which represents the watermark.

Another aspect of a method which conveys static information is the interference graph. An interference graph represents which variables in a method are live simultaneously. If two variables are live at the same time then an edge is drawn between them. This graph is used to facilitate register allocation for the method because two variables that are live simultaneously should not be assigned to the same register. Qu and Potkonjak [9] make use of the interference graph and the graph coloring problem to embed a watermark in the register allocation of an application. Graph coloring is an NP-complete problem for which many heuristic algorithms have been developed that work well for register allocation. The problem is defined as follows: Given a graph G(V,E) color the graph with as few colors as possible such that no two adjacent vertices are colored with the same color. To embed a watermark using the QP algorithm edges are added between chosen vertices in the graph based on the value of the message. The vertices are now connected and when the graph is colored the vertices will be highlighted with different colors yielding a new register allocation. So the algorithm embeds the watermark by adding fake interferences between variables forcing them to be assigned to different registers.

All of the previously described watermarking techniques are classified as static watermarking algorithms. The first dynamic watermark algorithm, CT, was proposed by Collberg et al. [3]. This technique builds a graph structure at runtime which is used to embed the watermark.

Of these proposed algorithms very little has been published on their implementation and evaluation. There are a few existing implementations of the CT algorithm, such as the one within the SandMark framework and that by Palsberg et al. [8]. A recent dissertation by Hachez on software protection tools [6] provides an implementation of the Stern algorithm. Other than these publications little work has been done in this area.

SandMark

SandMark [2, 10] is a research tool currently under development at the University of Arizona. The tool is designed for the study of software protection techniques such as code obfuscation, software watermarking, and tamper-proofing of Java bytecode. One of the goals of the project is to implement and evaluate all known software watermarking algorithms.

To aid in the evaluation of the watermarking algorithms a variety of tools have been incorporated into the system:

- An obfuscation tool which permits the study of algorithm resiliency by applying semantics preserving code transformations.
- A watermarking tool to study additive attacks in which an adversary adds a new watermark in an attempt to cast doubt on the original.
- A bytecode diff tool used to launch a collusive attack in which the adversary compares two differently watermarked programs.
- A static statistics tool to examine how well the watermark blends in with the code around it.

Through the use of the SandMark tool users are able to develop and test new watermarking algorithms. In addition, the tool is available for users to watermark any Java application which they have created and wish to distribute.

Conclusion

Platform independence and the use of the Internet are both valuable aspects of the computer industry. Unfortunately, they have made it easier for software pirates to illegally distribute software and for unscrupulous programmers to steal algorithmic secrets. Both of these issues are of ethical concern and have been the focus or recent research. Through this research a variety of prevention techniques have been developed using both hardware and software. Unfortunately, no single solution is currently strong enough to prevent piracy. However, through a combination of techniques, such as software watermarking, application developers can better protect their products.

References

1

Business Software Alliance. **http://www.bsa.org/**.

2

Collberg, C., Myles, G., and Huntwork, A. Sandmark -- A Tool for Software Protection Research. *IEEE Security and Privacy*, 1, 4 (July/August 2003), pp. 40-49.

3

Collberg C., and Thomborson, C. Software Watermarking: Models and Dynamic Embeddings. *Conference Record of POPL '99: The 26th ACM SIGPLAN-SIGACT Symposium on Priciples of Programming Languages*, 1999.

4

Davidson, R.L., and Myhrvold, N. Method and System for Generating and Auditing a Signature for a Computer Program. US Patent 5,559,884, Assignee: Microsoft Corporation, 1996.

5

Hachez, G. *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, 2003.

6

Low, D. Protecting Java Code Via Code Obfuscation. *ACM Crossroads*, 4, 3 (Spring 1998).

7

Monden, A., Iida, H., Matsumoto, K., Inoue, K., and Torii, K. A Practical method for Watermarking Java Programs. *Compsac 2000, 24th Computer Software and Applications Conference*, 2000.

8

Palsberg, J., Krishnaswamy, S., Kwon, M., Ma, D., Shao, Q., and Zhang, Y. Experiences with Software Watermarking. *Proceedings of ACSAC'00, 16th Annual Computer Security Applications Conference*, 2000.

9

Qu, G., and Potkonjak, M. Hiding Signatures in Graph Coloring Solutions. *Information Hiding*, 1999, pp. 308-316.

10

Sandmark. **http://www.cs.arizona.edu/ sandmark**.

11

Stern, J.P., Hachez, G., Koeune, F., and Quisquater, J. Robust Object Watermarking: Applications to Code. *Information Hiding*, 1999, pp. 368-378.

12

The Easter Egg Archive. **http://www.eeggs.com/**.

13

Venkatesan, R., Vazirani, V., and Sinha, S. A Graph Theoretic Approach to Software Watermarking. *4th International Information Hiding Workshop*, 2001.

Biography

Ginger Myles (**mylesg@cs.arizona.edu**) is a PhD student in the Computer Science Department at the University of Arizona under Christian Collberg. Her research interests include software protection, in particular software watermarking, and issues concerning privacy and security in ubiquitous computing environments. Currently, she is working on the SandMark project.