Dependency in Software Watermarking

D. Curran*, M. Ó Cinnéide, N.J. Hurley, G.C.M. Silvestre

Information Hiding Laboratory, Department of Computer Science,
University College Dublin, Belfield, Dublin 4, Ireland,
david.curran@ucd.ie

1 Introduction

The advent of bytecode languages such as Java and advances in decompilation tools have made it easier to infringe the copyright of software. This presents a serious challenge to the software industry, as modern Internet software is highly distributed and the potential for software piracy is immense. Software watermarking has been proposed as one means of protecting the intellectual property in software. The strategy is to embed secret ownership information (i.e., a 'watermark') in a program that cannot be easily removed by unauthorised parties but that can be reliably extracted by an authorised decoder.

One of the most effective watermarking techniques proposed to date is the dynamic graph watermarking scheme of Collberg et al [1, 2] which is resistant to attacks such as optimisation and obfuscation. In this method a watermark is represented by a number that is encoded in the topology of a particular graph structure. The program to be watermarked is altered so that this graph is formed in the structures created by the program running for a particular input. The watermark detector searches through the heap of the program for a graph of required type and once located, the number is decoded from the graph.

For such an embedding to take place, the program text must be augmented with *watermark generating code*. Such code does not represent the watermark itself, but rather is a set of instructions that result in the watermark being generated in a dynamic state of the running program.

The goal of this paper is to investigate a practical implementation of dynamic graph watermarking, focusing particularly on embedding stealthy watermark generating code in the program's text. Our scheme is based on the premise that watermarking code should not be distinguishable from ordinary program code. This is because one strategy for an attacker whose goal is to destroy the watermark is to attempt to directly locate the watermark generating code within the program text and remove or alter it.

2 A Blended Dynamic Graph Scheme

We aim to embed the watermark generating code deeply into the structure of the host program so that not only is detection of the watermark code difficult, it is also difficult to change the watermark code without also changing the observable run-time behaviour of the host program.

As a first step towards this goal we focus on reducing the number of statements that need to be added in order to produce the graph. By reducing the resource usage of the watermark generating code we make it more difficult to detect. This resource usage can be measured in added lines of code.

*Supported by Enterprise Ireland Basic Research Grant SC/2002/178

0-7803-8482-2/04/$20.00 ©2004 IEEE.

Our strategy for reducing this number of lines of code is to reuse watermark generating code segments to build isomorphic subgraphs each time they are executed. This allows us to produce N identical graph parts using a code segment that is executed at least N times. As a consequence of code blending the watermark generating code is tied in with the control flow of the program which further enmeshes watermark generating code with the host program.

Software engineering principles state that data sharing inside a program should be kept to a minimum. Violation of this principle makes watermark code conspicuous and thus open to attack. A large watermark generated from many code segments disobeys this data sharing principle. We reduce the data dependency by representing the watermark using a set of smaller graphs rather than a single graph. The watermark number is encoded by a number of smaller graphs. The detector recreates the original graph using placement information encoded in each of these graph enumerations. This allows us to create watermarks in separated segments of code without these segments needing to share information.

3 Results

We examine if our system increases the efficiency of watermark generation and we compare our system against the Sandmark implementation [1] to examine whether the unusual properties of programs watermarked with Sandmark render them susceptible to attack.

Our tests showed that using our current code blending strategy reduces the number of watermark generating statements by an average of 13%. This shows that by blending watermark generating code with the host program more efficient watermarks can be achieved. Our scheme was found to be resistant to optimisation and obfuscation attacks.

We created a byte code analyser to detect Sandmark's watermark generating code. Each class in the watermarked program was evaluated using a metric which calculated a weighted sum of a set of characteristics that are considered rare in ordinary classes, but which appear in the Sandmark watermark generating class. Our static analysis attack successfully removes calls to Sandmark's watermark generating code even when the code is obfuscated.

4 Conclusion

Our experimental results show that if watermark generating code is distinguishable from ordinary program code the watermark can be removed. On the other hand, by obeying the principles of software engineering and by blending watermark generating code with the host program software watermarks become less susceptible to attack. By seamlessly integrating the watermarking code with the host program we have increased the robustness of software watermarking to attacks based on the static analysis of class files.

References

[1] Christian Collberg, SandMark: A Tool for the Study of Software Protection Algorithms. *http://www.cs.arizona.edu/sandmark/*, 2004.

[2] C. Collberg and C. Thomborson,Software watermarking: Models and dynamic embeddings. *Proceedings of POPL'99 of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 311–324, March 1999.

0-7803-8482-2/04/$20.00 ©2004 IEEE.