STATIC SOFTWARE WATERMARKING

Claudiu Chiru, Spiru Haret University Constanta
E-mail claudiu.chiru@seanet.ro

Abstract

Watermarking is a technique for adding a message to data to identify the owner of the data. In this paper we propose a method to watermark software by a static method.

Key words: software watermarking, static watermarking, opaque predicate

1. INTRODUCTION

Appearance of Internet represented a true leap referring to the inter-human communication, offering new ways to make business, access to electronic libraries, electronic newspapers. The possibilities are enormous. The information can freely move in this immense network. Still exist problems concerning protection of the multimedia information and to the copyright. The cryptography can offer a wide scale of security services, but once legally entered in the possession of a multimedia document, later on can be created copies, and using the same network (Internet), the material can be spread very quickly. The watermarking technique offers a solution for the document's protection (images, sounds, texts), so that it can be verified the document's authenticity. By copying, the information with regard to the document's license will be transmitted to the copy. In the verification's moment, the license information is extracting, and so can prove the property rights referring to the document. In English, watermark has the meaning of filigree.

1.1. Definition of Watermark

The watermark is a signal that can be added to the digital data (video, audio, static images) which can be detected or extracted later. The extracted signal can offer some information about data that contained it.

1.2. Marking the Software

The watermarking technique can be extended to software. Colberg, in his paper (Colberg 1999) refers to two types of watermarks: static and dynamic watermark. The static watermarks are embedded in string sections, debugging information sections or in the code section. Dynamic watermarks are stored in the program's execution state.

2. STATIC MARKING OF THE C/C++ PROGRAMS

This method proposes to mark the C programs by inserting a code sequence, which doesn't execute. This marking method uses an opaque predicate that is hard to evaluate.

2.1. Marking by Inserting of a New Function in the Program

Starting from Monden et al. paper (Monden et al. 2000) the marking method consists in the introducing of a function in a C / C++ program, that will never be executed. The fact that the function will not be executed is ensured by the existence of an opaque predicate, with a true/or false value, always.

```
.
.
if(PT)
{
        func(p,q,r,…)
}
```

p^T can be a predicate that results from an equation with integer coefficients with solutions in the integer numbers set. Also, p^T can be the negation of a false predicate. p^F can be selected from the set of the inequalities that have integer numbers solutions.

After program's compilation, the function's body is localized and then changes are made in structure of the function code, according to the watermark which is entered. The program's execution will be followed normally, because the changes will not affect the program (function doesn't execute).

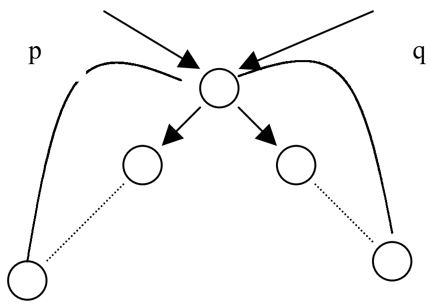

Figure 1. Graph in the form of binary tree with two circular lists

The method is vulnerable at the following attacks:

- The code's re-ordering. Certain instruction sequences can be executed in another order, if the result of the instruction sequences is not depending on their succession. By code's re-ordering, the message will become unintelligible.
- The code's decompilation and compilation. The obtained code will be functional, but the watermark message can not be extracted.
- The determination of the place where is inserted function and its elimination from program. This can be made using an analyzing program, which can generate the program's execution flow. So, after more executions, one can remark that the function never executes, hence it is a "dead" code, which can be eliminated from program. Selecting of an opaque predicate (which can have the true or false values) prevents this kind of attacks. A source of opaque predicates is the utilization of the expressions with pointers.

```
v=random(m)  /*  random  number
generation    in    the    interval
[0,m], m>n */
/* n is the number of vertices
in  the  left  or  right  subtree
*/
p=q=root;/*  root  –  pointer  to
the root */
for(i=1;i<=v;i++)
{
  p=p->left;
  q=q->right;
}
…
if(p==q)
  f(…);
…
```

Figure. 2. Example of inserting a function with an opaque predicate

The graph from figure 1 is a tree in which the last node from each sub-tree has a pointer towards root.

Number of vertices from the left sub-tree is equal with that from the right sub-tree.

By inserting of the graph in program, is possible that at a certain moment, a predicate p ! = q to have the true value, and at other time moment, p ! = q to have the false value. In this situation, the instruction

```
if (p ! = q)
     f (a, b, c, ...)
```

has, as probable result, the execution of the "f" function. In such conditions, "f" must be designed in a way that it doesn't modify the variables from program and, in the same time, to incorporate the static watermark.

The f function will be executed with a probability equal to $\frac{1}{n^2}$. So, the code sequence that contains the watermark will be executed with the same probability and the f function will use only local variables.

The structure of such a function might be:

```
void f(int a,int b,int c,…)
{int d;
int f[10];
d=a+b+c; /* Identifying
sequence */
/* Inserting position for the
watermark  */
f[0]=1;f[1]=1;f[2]=1;…;f[9]=1;
...
return;
}
```

After the compilation phase the new watermark can be inserted by replacing the identification sequence and the watermark support `(f[0],f[1],...)`. The assembly code is:

```
mov ax,[bp+4]   ⎫   function
add ax,[bp+6]   ⎬   identification
add ax,[bp+8]   ⎪   sequence
mov si,ax       ⎭

mov word ptr [bp-14],0001 ⎫
mov word ptr [bp-12],0001 ⎪   Code
mov word ptr [bp-10],0001 ⎪   sequence
mov word ptr [bp-0e],0001 ⎬   which is to
mov word ptr [bp-0c],0001 ⎪   be replaced
        …                 ⎪
mov word ptr [bp-02],0001 ⎭
```

The program must be equivalent from the functionality's point of view. So, the replacement will take place only in the source operand area from the mov instructions. Code size depends on watermark size. For example, a two byte watermark can be inserted in a f[0]=1 instruction. For an n byte watermark approximately n/2 instructions will be necessary. The number of instructions can be reduced to half if we can use 32 bits variables such as unsigned long.

3. CONCLUSIONS

Above-mentioned methods represent viable solutions for the program protection against unauthorized utilization. They don't stop a malevolent user to use a program purchased by illegal ways. On the other hand, if necessary, can be proved the fact that the user is not the right owner of the program.

4. REFERENCES

(Colberg 1999) Christian Collberg - Software watermarking: Models and Dynamic Embeddings POPL '99 , Proceedings.

(Monden et al. 2000) Akito Monden, Hajimu Iida, Ken-ichi Matsumoto, A Practical Method for Watermarking Java Programs, The 24th Computer Software and Application Conference, Taipei, Taiwan, Oct 2000.