Embed digital watermarks in executable program memory

S V Belim¹², S N Munko¹ and S Yu Belim¹
¹Omsk State Technical University, 11, Mira, Omsk, 644050, Russia
²Siberian State Automobile and Highway University, Omsk, Russia

E-mail: sbelim@mail.ru

Abstract. The article proposes a method for embedding digital watermarks into executable program code. A digital watermark is generated in the process RAM. Existing information structures, such as dynamic arrays and objects, are used to accommodate parts of the embedded message. The message embedding program modifies the size of the data structures. The digital watermark is placed in additional memory locations. Two approaches to the placement of digital watermarks are proposed. Arrays are ordered in lexicographic order. Parts of the embedded message are placed in arrays. There are no array names in the executable code. Information for steganographic analysis is not available. Arrays for embedding can also be selected in any order. Portions of the digital watermark are written to a dynamic unidirectional list. The unidirectional list is embedded in arrays. Embedding into objects of the same class is carried out using similar methods.

1. Introduction

Currently, controlling the distribution of software is a topical issue. This problem is related to monitoring programs unauthorized copying. Changes are made to programs during illegal distribution to destroy information about the license key. Algorithms for embedding hidden information are one solution to this problem. A hidden message helps track and identify illegal copies of programs. Hidden information in source code fragments allows you to check the illegal use of other people’s program code fragments. Methods for embedding hidden messages in various objects are called steganography. Traditional methods of steganography use various digital objects as a container: images [1-5], audio files [6-8] or video files [9-12]. Algorithms for embedding messages into network protocols have become common in recent years [13-15].

Digital watermarks are steganographic messages embedded in an object that contain information about it. Digital watermarks are used to protect intellectual property on the objects in which they are embedded.

Algorithms for embedding digital watermarks in program code encounter several problems. These problems are not present in similar tasks for image or video. The first problem is the selection the modifiable data. The container data changes when embedded. These distortions are small. The modified image is visually little different from the original image. This approach is not applicable when embedding hidden messages into an executable program. Any program is presented as data and executable commands for processing this data. Any change in the command code leads to incorrect operation of the executable code as a whole. The corrected program may not be compiled. Changing data is also not allowed. Any change to the constants that initialize the variables causes the program to run incorrectly. Some articles have proposed algorithms to solve this problem. In article [16], the authors propose to embed messages in NOP-chains present in any executable code. NOP-chains appear in the
program when the code segments are aligned to the size of the memory pages. The authors suggest replacing NOP commands with some commands containing an embedded message. These commands should not make changes to the program and prevent compilation. Retrieving this embedding message is performed by disassembling the executable code. Information about the location of the embedding message in the program code is required to extract it. This approach is not applicable when embedding messages into program code in a high-level language. The second approach proposed in article [17-26] uses the objects of the program to embed the message. This approach is not universal. It is not applicable to console applications and dynamic libraries.

The second problem of embedding digital watermarks into program code is that the program algorithm has a logical structure. The program structure facilitates the task of finding and eliminating the embedding message. Steganographic analysis of images does not correlate between data. Some geometric shapes are present in the image. Finding shapes in an arbitrary image is a difficult task to recognize images. A message embedded in the least significant bit of the blue component with low container occupancy and random pixel selection is not detectable. The logical structure of the program can be restored to the original or disassembled code. Analysis the logic of the program code allows you to easily identify non-functional inserts and detect embedding messages. The data present in the program are constants for initializing variables. Constants have a small volume. Modification of constants is not allowed.

A hidden message must be inseparable from the container. This is the third problem with embedding hidden messages into program code. Deleting a message should destroy the container. The hidden message must be closely related to the program code.

Program code has one important advantage over other containers for hiding a message. This advantage is the automatic stability of digital watermarks. The stability property is the ability to retrieve hidden information with small modifications of the container. For executable code, the only acceptable modification is the variation of external data. The stability of the digital watermark is ensured by the lack of binding to the external data stream.

This paper proposes a new approach to the generation of digital watermarks in executable program code at the stage of its development in a high-level language.

2. Description of the system
We formulate the basic requirements for embedding a digital watermark into executable program code.

1. The digital watermark is not present in the source code as a value of some constant. Static analysis of source code or disassembled code easily detects such characters.
2. Digital watermark is generated dynamically during program execution.
3. A digital watermark is generated at some stage of the program execution. The step for appearing a digital watermark generation program is secret information.
4. The digital watermark is destroyed at some point in the execution of the program. A digital watermark exists in the RAM for a time interval. At this time interval, the digital watermark can be extracted. This approach significantly complicates dynamic code analysis to detect a digital watermark.
5. The digital watermark depends on the key information. Key information is entered interactively by the user.
6. The digital watermark generation instructions are distributed throughout the source code. This requirement prevents the removal of the digital watermark from the source code.
7. A digital watermark is generated in RAM. The digital watermark must not occupy a continuous block of address space.
8. The digital watermark is divided into blocks. These blocks are stored using some dynamic data structure. The block locations depend on the key information entered by the user.

To fulfill these requirements, it is necessary to develop three algorithms: an algorithm for forming a digital watermark during the execution of the program, an algorithm for placing a digital watermark in RAM, and an algorithm for extracting a digital watermark.

Placing a digital watermark in RAM is usually done by allocating additional memory and writing information to it. The allocation of independent memory under the digital watermark can be detected using API tracking of program calls. To allocate memory, the program must access the operating system.
If a digital watermark is stored in separate memory locations, it can be detected by sequentially analyzing all allocated memory areas. The addresses of the allocated memory areas are determined when system calls are intercepted. Sequential modification of the values in memory cells allows to determine which of the cells contain information important for the program operation, and which memory cells store a digital watermark that does not participate in the program operation. To prevent this attack, the memory under the digital watermark must be allocated simultaneously with the allocation of memory for the structures used by the program to solve basic problems.

In this work, embedding is carried out at the stage of program development in a high-level language. In this case, the allocation the memory for the digital watermark can be accomplished by increasing the size of the memory areas allocated for the main variables. With this approach, the location of the digital watermark cannot be detected when analyzing system memory allocation calls. The expansion of the memory area for storing the digital watermark can be accomplished in one of three ways.

1. Increases the size of dynamic arrays.
2. Add new fields to existing classes.
3. Increases the size of memory cells allocated to variables.

In the first case, a number of cells for storing the digital watermark are added to the existing array. If such an array is one, then the digital watermark is completely written to these memory locations. If there are multiple arrays, the digital watermark is distributed among them. The distribution of the digital watermark parts between several arrays can be organized in various ways. Digital watermark $w$ can be broken into parts $w_1, \ldots, w_n$.

$$w = w_1 || w_2 || \ldots || w_n.$$  

The digital watermark parts $w_i$ ($i = 1, \ldots, n$) are distributed among the additional cells of the arrays in lexicographic order (Figure 1). One type of variable is selected to eliminate uncertainty. Embed into all arrays of this type. The number of array elements added is determined by the digital watermark size. Digital sign splitting can be done equal shares. In this case, the implementation of embedding is simplified. Searching for a digital watermark is also simplified. Digital watermark can be split into unequal fractions. This approach significantly complicates steganographic analysis. The digital watermark embedding program must analyze the source code of the program, search for arrays of a given type and increase their size. The size is increased by changing the pattern size parameters. The embed program must remember the names of the arrays, the original number of elements in them, and the number of elements added to them. Thereafter, the program generates an embedding list and embedded information.

The order for the digital watermark portions may be recorded in a new structure. Dynamic memory should be allocated additionally for such structure. This approach is contrary to our system requirements.

In the second case, the order of embedding information into additional array elements is random. To do this, we need to use some dynamic data structure. In this work, a unidirectional dynamic list is used (Figure 2). The elements of the unidirectional dynamic list are placed in additional array cells in an arbitrary order. Three challenges need to be addressed in this approach. The pointer to the first dynamic list item must be created. References for navigating between list items must be configured. List items must be converted to an array format and converted back. This approach is much more difficult to steganographic analysis. The reference to the first element can be organized as a new variable. ut this approach does not satisfy our requirements. The new variable requires additional memory allocation. The first or last array in lexicographic order can be selected for the first item in the list. The dynamic list approach reduces the amount of embedded digital watermark and requires a large amount of memory for overhead information.
Figure 1. Portions of the digital watermark are distributed among arrays in lexicographic order.

Figure 2. Portions of the digital watermark are distributed among arrays based on the list.
When creating additional fields in existing classes, the same approaches can be used as for arrays. Portions of the digital watermark may be distributed among objects of the same class in lexicographic order (Figure 3). A dynamic list may be used to place portions of a digital watermark between objects of the same class (Figure 4).

**Figure 3.** Parts of the digital watermark are distributed among objects of the same class in lexicographic order.

**Figure 4.** Portions of the digital watermark are distributed among objects of the same class based on the list.
The task of embedding a digital watermark by increasing the cell size of individual variables is more complex. The technology consists in increasing the size of the memory cell allocated for the variable. This approach faces many challenges. It requires overriding arithmetic operations. We must also monitor the overflow of memory cells. In a normal situation, overflow results in loss the most significant bits. If we embed digital watermarks, overflow can modify the embedded data.

The extraction of the digital watermark must be done by the program itself. Retrieving a hidden message by a third-party program requires reading from a foreign address space. This operation is not allowed by the operating system. Some event should trigger the program code portion to retrieve the digital watermark. This event may be the pressing the set of keys. The second option is to enter a passphrase in some window. The digital watermark output event must be a secret event. In both cases, some subprogram should be activated to output the digital watermark. This subprogram must have three properties.

1. The subprogram must have information about the placement of a digital watermark in memory,
2. The subprogram should be generated automatically at the step of embedding the digital watermark.
3. The subprogram should not be distinguished against the background of the rest of the program in its style and structure.

The implementation of the system for automatic generation and embedding such a subprogram into the program source code has a technical difficulty.

3. Conclusion
We have proposed algorithms for embedding a digital watermark into an executable program using existing data structures. The proposed algorithms do not create new data structures. Digital watermarks cannot be detected on API memory allocation calls. The inventive scheme creates digital watermarks that are difficult to detect in process memory analysis. The main vulnerability of the digital watermark embedding scheme is the digital watermark extraction routine.

References
[1] Mazumder J A, Hemachandran K 2014 International Journal of Computer Sciences and Engineering 2(7) 90
[2] Laskar S A, Hemachandran K A 2014 IJERT 1(3) 3400
[3] Qureshi M A 2012 International Journal of Computer and Electrical Engineering 4(4) 558
[4] Saurabh R M 2014 IJAEAS 1 2348
[5] Shinde M 2015 IJRASET 3 345
[6] Senbin Y 2010 Journal of Multimedia 2(5) 151
[7] Singh J 2009 International Journal of Electronics and Communication Engineering 3(11) 2207
[8] Tiwari A 2014 IJEIT 3(7) 121
[9] Manekandan G R S, Rajkumar F V 2012 International Journal of Comp. Appl. 41(18) 24
[10] Chandrika Y J S, Kalyani D, Lavanya P, Kavitha T 2016 International Research Journal of Engineering and Technology 3(7) 368
[11] Sujatha C N 2015 IJSETR 4(2) 245
[12] Belim S V, Cherepanov P G 2019 Journal of Physics: Conf. Series. 1260 022001
[13] Gutierrez-Cardenas J M 2017 International Journal of Security and Its Applications 11(4) 71
[14] Janicki A, Karas M, Mazurczyk W, Szczypiorski K 2016 Multimedia Tools and Applications 75(21) 13521
[15] Karas M, Mazurczyk W, Szczypiorski K 2014 International Journal of Computers, Communications & Control 8(3) 432
[16] Gabrys R, Martinez L, Fugate S 2020 HotSoS '20 25
[17] Lu K, Xiong S, Gao D 2014 CODASPY '14 265
[18] Tian Z, Yang H 2021 J Image Video Proc. 2.
[19] Zashcholkin K, Drozd O, Shaporin R, Ivanova O, Sulima Y 2019 IEEE East-West Design & Test Symposium (EWDTS) 1
[20] Zashcholkin K, Drozd O, Ivanova O, Bykovyy P 2020 CEUR Workshop Proceedings 2623 267
[21] Zashcholkin K, Drozd O, Sachenko O, Ivanova O, Boliubash Y 2019 IDAACS’19 234
[22] Zashcholkin K, Drozd O, Ivanova O, Shaporin R, Veselska O, Stepova H 2019 DESSERT’19 175
[23] Jin C, Pan M, Zhang D 2010 Int. Con. of Information Science and Management Engineering 35
[24] Jian-qi Z, Yan-heng L, Ke Y, Ke-xin Y 2009 WRI World Congress on Computer Science and Information Engineering 602
[25] Monden A, Iida H, Matsumoto K, Inoue K, Torii K 2000 COMPSAC2000 191
[26] Collberg C, Thomborson C 1999 POPL ’99 311