Tian and Yang *EURASIP Journal on Image and Video Processing* (2021) 2021:2
https://doi.org/10.1186/s13640-020-00541-3

EURASIP Journal on Image and Video Processing

RESEARCH Open Access

Code fusion information-hiding algorithm based on PE file function migration

Zuwei Tian* and Hengfu Yang

* Correspondence: 35568625@qq.com
School of Information Science and Engineering, Hunan First Normal University, 410205 Changsha, China

Abstract

PE (portable executable) file has the characteristics of diversity, uncertainty of file size, complexity of file structure, and singleness of file format, which make it easy to be a carrier of information hiding, especially for that of large hiding capacity. This paper proposes an information-hiding algorithm based on PE file function migration, which utilizes disassembly engine to disassemble code section of PE file, processes function recognition, and shifts the whole codes of system or user-defined functions to the last section of PE file. Then it hides information in the original code space. The hidden information is combined with the main functions of the PE file, and the hidden information is coupled with the key codes of the program, which further enhances the concealment performance and anti-attack capability of the system.

Keywords: Information hiding, PE file, Function migration, Code fusion

1 Introduction

PE file is a standard format for executable file in Windows environment, which is one of the most important software formats in the Internet. The code section is the most important section in the PE file, which is used to store the executable instruction codes, including user-defined function code and static link library function code, which is the main part of the PE file. Combining hidden information with program instruction code can effectively improve the concealment of information hiding algorithm based on executable file.

At present, the PE-based information-hiding algorithms are divided into the following three categories: One is the information hiding method based on the PE file redundant space [1–20]. The second is the information hiding method based on PE file data resources [21–23], the third is the information-hiding method based on PE file import table [24–28]. The existing PE file hiding algorithms mainly exist the following shortcomings: First, the redundant space of PE files is open to people familiar with the PE file format, and there are powerful PE file analysis tools on the market, such as Stud_ PE and PE Explorer Lord PE. Obviously, because of the use of the redundant space inherent in PE files for information hiding, security is not good. The second is that the hidden space is too concentrated, the hidden information is easily exposed, and the concealment is poor. The third is the structure of the PE file is transparent; the use of

PE files structure characteristics to hide information, once the transformation of its structural characteristics, hidden information will be destroyed. Fourth, the hidden information is not combined with the program function, there is no close association with the program itself, the hidden information and program instruction code is low coupling, the ability to resist deletion, modification, filing, and other attacks is poor [29–31].

This paper proposes a highly concealed information-hiding algorithm based on the migration of PE file code section functions, which enhances concealment and improve the system's anti-attack capability, on the basis of fully analyzing the characteristics of PE file code section. First, through the disassembling engine disassemble PE file code section, the function recognition algorithm is used to identify the standard functions in the program, and then the function modules in the program are migrated to the redundant space or the last section of the code section. In this way, the hidden information and program instruction code closely combined, greatly improve the system's concealment and anti-attack ability.

The rest of the paper is organized as follows: In section 2 analyses the structure of the PE file code section. Section 3 describes the proposed method. Section 4 is experimental results and discussion. Finally, section 5 summarizes the paper.

2 PE file code section anatomy

2.1 Key data structures of code section

The section name of the code section in the PE file is generally .code (or .txt, which is related to the compiler), and its property value is 0x60000020, indicates that the section is executable and readable, and contains instruction codes, which is generally located next to the section table. It is the first section of the PE file, in front of other sections. The data structure related to the code section is VirtualSize field, VirtualAddress field, PointerToRawData field, and SizeOfRawData field.

When the PE file is loaded, the loader of the Windows operating system continuously reads SizeOfRawData bytes of data, the entire section of code data, from the PE file offset to the position of PointerToRawData, and maps it to memory.

2.2 Data organization of code section

The field value PointerToRawData in the code section table indicates the offset address of the code section in the disk file, and SizeOfRawData indicates the amount of disk space taken up by the code section after the file is aligned, as the disk file space is generally aligned by 512 bytes. The space that the code section actually occupies (the value of SizeOfRawData) is larger than the value of VirtualSize (unaligned), thus creating the redundant space of the code section, and the difference between the value of SizeOfRawData and VirtualSize is the size of the redundant space of the code section. In the code section, the redundant space is filled with 0. Setting S_r as the size of the redundant space for the section:

$$S_r = \mathrm{SizeOfRawData} - \mathrm{VirtualSize} \tag{2.1}$$

Figure 1 is the code section structure of the PE file generated by the VC++ compiler.

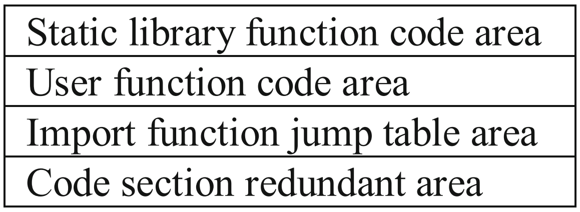
Static library function code area
User function code area
Import function jump table area
Code section redundant area

Fig. 1 Code section structure of PE file

2.3 Determination of the entry address of program

In general, when the PE file is loaded by the Windows operating system loader, the code section is loaded onto the 0x00401000 address (the value of ImageBase plus the value of VirtualAddress), AddressEntryOfPoint of the HEADER32 structure indicates the address of the program executable code entry point, that is, the RVA of the first instruction to be executed in the PE file, and its value plus the base address in the PE file memory, is the starting virtual address of the entry function of the program when it runs. For example, the AddressEntryEntryPoint value of a PE file is 0x0001120D, the entry address of the program is 0x0041120D. Some of the programs that insert code into PE files are to modify the address here to point to their own code, and then jump back after being executed.

The IAT is located before the module entry point in the .text segment (IAT table is actually a collection of jump instructions). When the Windows loader loads the executable program into the address space of the process, the actual memory address for each import function is determined, and the IAT table is also determined.

3 Proposed method

In this section, we depict the proposed information-hiding scheme using PE file function migration, and then it hides information in the original code space.

There is usually at least a code section in a PE file, which holds executable code. The function of a program is achieved by executing instructions in a code section. Therefore, hiding information in the PE file code section by combining hidden information with instruction code, which can effectively improve concealment and anti-offensive. But directly hiding the information in the PE file code section, some of the hidden information will be converted into some extremely abnormal instructions when being disassembled, which is easy to arouse the suspicion of attackers. In order to improve the resistance to disassembly and other reverse analysis tools, we will convert hidden information to instructions, disguised as a function (functionalization), embedded into the code section. The hidden information and PE file executable code are integrated, which improves the concealment. At the same time, in order to solve the problem of over-concentration of hidden information, a method of migrating one or more functionally independent modules (functions) in the executable code of PE files to redundant spaces in the code section is proposed, and the information is hidden between the normal function instruction code, so that the hidden information and PE file executable code are closely integrated. It further enhances the concealment and security of the system.

3.1 Disassembly algorithm

The principle of disassembly software is to first identify the format of the executable file, distinguish the code and data, determine the file offset address at the entry point of the code section, then utilize the knowledge of lexical analysis and grammar analysis to analyze, decode according to the instruction format of the X86 architecture, and finally output the corresponding assembly instructions.

Disassembly technology can be divided into static disassembly and dynamic disassembly. Static disassembly refers to the conversion of the target program into the corresponding assembly language program without executing the target program. Dynamic disassembly refers to tracking the execution of the target program, in the process of execution disassembling the target program. One advantage of static disassembly is that the entire target program can be processed at once, while dynamic disassembling can only handle the parts to which the target program is executed. Currently, the commonly used disassembly tool software are IDA Pro, Ollydbg, Win32Dasm, SoftICE, Windbg, etc.

3.2 Design and implementation of disassembler

The role of the disassembling engine is to translate machine codes into assembly instructions. Developing an excellent disassembly engine requires an in-depth understanding of machine instruction coding for Intel's X86 architecture, with a long development cycle. Common open-source disassembling engines are udis86, Proview, ade, xde, etc. [28]. OllyDbg's own disassembling engine is also relatively powerful, but its instruction set is incomplete and does not support MMX and SSE well.

We use Udis86 to build a disassembler, the main steps of which are as follows:

Step 1: Deploy the code and header files of the Udis86 disassembling engine to the system or directly into the project, refer to the "udis86.h" header file.
Step 2: Define an Udis86 object (ud_t ud_obj), set disassembly mode to 32 bits, set the instruction format for intel instruction format, set the start address of the first instruction, set the input source, which can be memory, or use ud_set_input_ file is set directly to file input and other initialization work.
Step 3: Looping, disassembling all the instructions in the input source.
Step 4: To carry out instruction analysis.
Step 5: Record the results of instruction analysis.

The result of disassembly is the same as that of OllyDbg using the built disassembler to disassemble the writing board program of the system (write.exe). The high-quality disassembler is the basis for further function recognition.

3.3 Function identification and location

In the process of application programming, modular programming is usually adopted. According to the top-down method, the program is broken down into many functional independent modules, each independent module is implemented by a function. In order to implement some complex functions, a large number of library functions are provided by the system, including static link library functions and dynamic link library functions. Static link library functions include system library functions and dedicated library functions. During compiling and linking, just like the user-defined function, the code will be linked to the target code of the executable program. For the dynamic link library function called, the target code is not in the executable file but in a DLL file. According to statistics, library function code accounts for an average of 50-90% of the target code in programs written by advanced languages [32].

In order to further improve concealment and integrate hidden information with the program's key function code, we propose an information-hiding algorithm for migrating function code, and the recognition and location of functions is the basis for this algorithm.

After disassembling the target program code section, according to the compilation principle and the specification of the function call, the starting address of the function module is generally the value of the address expression after the CALL instruction, that is, if there is an instruction CALL ADDR in the assembly code, there must be a function module with ADDR as the starting address. The function module ends with the RET instruction. Since there may be multiple exits in the function module (multiple RET instructions), according to the characteristics of the function, the end address of the function can be determined by the following algorithm:

Function: Determine the end address of function module.

Input: Starting address of the function module (F_begin).

Output: End address of the function module (F_end).

```
    The algorithm is described as follows:
1. begin
2. F_temp=F_begin
3. Find address of the first RET instruction from F_temp.
4. F_end=F_ret
5. if from F_begin to F_end without JMP instruction then
    return F_end
6.   else find F_jmp (JMP with maximum address) from
          F_begin to F_end
7. if   F_jmp≤F_end
       return F_end
8. F_temp= F_jmp
9. goto 3
10. end
```

Through the address expression after CALL instruction and the above algorithm, the start address and the end address of a function module can be determined, and the length of the function instruction code can be calculated.

Using this algorithm to test the function of notepad.exe and thunder program thunder.exe, the experiment shows that 59 functions can be effectively identified from "notepad.exe" program, and 4086 functions can be effectively identified from "thunder.exe" program. It can meet the needs of migration function well (Table 1). Because the purpose of function recognition in our system is to migrate function to implement information hiding, we have simplified the function recognition algorithm, for some special functions and functions with short code, will be ignored in the algorithm, which does not affect the effectiveness of the algorithm and information hiding. If the amount of information to hide is large, you can hide the information in the extended function area by extending the length of the migrated function, or, after the information to be hidden is functionalized, stored in the last section of the PE file, scattered between the two migrated functions.

3.4 Function migration

In order to closely combine the hidden information with the instruction code of the executable, we propose an information-hiding algorithm for function migration that hides the information in the storage area of the original function module by migrating the function module in the target program to the last section. Function recognition is the basis for function migration, locating function by function recognition, and determining the file addresses of function (including function start address and end address) in disk file, relative virtual addresses and length of function, then correcting the relevant instructions in the function module and overwriting the relevant property values of the PE file. The migration of functions can be implemented. Because the target program code section at the time of the link holds the static library function code that is called first, followed by the code for the user-defined function. In order to improve concealment, user-defined functions are preferred when selecting the migrated functions.

Table 1 Example of function recognition

Filename	File size (bytes)	Function sequence number	Function size (bytes)	Function address range	Relative virtual address range
notepad.exe	5632	1	126	D5D-DDA	195D-19DA
		2	85	0FE8-103C	1BE8-1C3C
		3	394	1173-12FC	1D73-1EFC
		4	346	36F9-3852	42F9-445A
		5	496	4D49-4F38	594C-5B3B
		…	…	…	…
thunder.exe	1808176	1	203	1C36-1D00	1C36-1D00
		2	152	1D01-1D98	1D01-1D98
		3	1648	21D5-2844	21D5-2844
		4	206	2FE4-30B1	2FE4-30B1
		5	218	359F-3678	359F-3678
		…	…	…	…

Let $\mathrm{OFFSET_{old}}$ and $\mathrm{OFFSET_{new}}$ represent original offset and new offset of the call instruction, respectively. $\mathrm{RVA_{old}}$ indicates the relative virtual address before the CALL instruction being migrated. $\mathrm{RVA_{new}}$ indicates the relative virtual address after the being migrated. $\mathrm{SECTION_{old}}$ and $\mathrm{SECTION_{new}}$ represent the actual size of the section before and after the migrated function (VirtualSize), and len(P) represents the length of the P function instruction code, respectively.

The main steps of the function migration algorithm are as follows:

Step 1: The function is located by the function recognition algorithm, and the selected function module to be migrated is read into memory.
Step 2: Locate at the end of the last section (the PointerToRawData value of the last section plus the value of the actual size of the last section, VirtualSize), write the starting address of the function to be migrated (located through that address when extracting information), and then write the instruction codes of the function to be migrated.
Step 3: Fix the address value in the CALL instruction inside the function after being migrated.

$$\mathrm{OFFSET_{new}} = \mathrm{OFFSET_{old}} + \mathrm{RVA_{old}} - \mathrm{RVA_{new}} \tag{3.1}$$

Step 4: Fix the size of the section and align the SizeOfRawData value of the section by FileAlignment.

$$\mathrm{SECTION_{newsize}} = \mathrm{SECTION_{oldsize}} + \mathrm{len(P)} + 4 \tag{3.2}$$

Step 5: Fix the PE file mirror size, mirror size is aligned according to the value of SectionAlignment.
Step 6: Change the section property to executable.
Step 7: Set the relocation table size to 0.
Step 8: Write a jump instruction at the beginning of the original function, which jumps to the start address of the migrated function.

It ensures that the function migration does not affect the function of the program through function migration and modifying the migrated function, so that the area occupied by the original function module can be used for information hiding, and the hidden information is tightly coupled with the key code of the executable program, which can effectively improve the concealment and security of the system (Fig. 2).

3.5 Information-hiding algorithm

After function recognition and function migration are completed, the information hiding is relatively simple, and its main steps are described below:

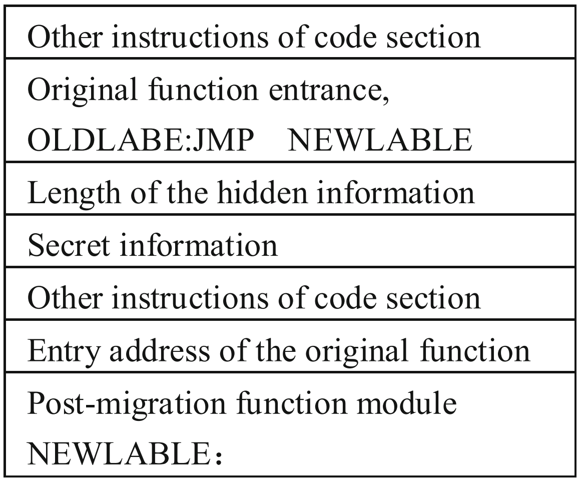
Other instructions of code section
Original function entrance,
OLDLABE:JMP NEWLABLE
Length of the hidden information
Secret information
Other instructions of code section
Entry address of the original function
Post-migration function module
NEWLABLE：

Fig. 2 Example of information hiding by function migration

Input: Original carrier PE file P, information to be hidden M, public key pk.
Output: Hidden PE P′ file.

Step 1: Using the public key pk and asymmetric encryption algorithm RSA, the hidden information M is encrypted and the encrypted information M' = Encrypt (pk, M) is obtained.
Step 2: The code section of the original carrier PE file P is disassembled by using the disassembling engine.
Step 3: Use the function recognition algorithm to recognize function of the assembly code produced by step 2, record the start address and end address, length of each identified function, count the sum of the number of functions and function lengths of all the identified functions, and sort the number by the size of the function by the starting address.
Step 4: According to the length of the information to be hidden, move a function from small to large of function number to the end of the last section, and write the first 4 bytes of the start address of the function module to the original function address after migration.
Step 5: Write a jump instruction at the beginning of the original function, jump to the beginning of the migrated function, and then write the length of the hidden information and the hidden information.
Step 6: Determine whether the information is all hidden, if it is to turn to step 7, otherwise turn to step 4 repeat the same operation.
Step 7: Modify the size of the PE file section, the size of the mirror, and change the properties of the section to be executable.

3.6 Information extraction algorithm

Input: A PE file P' with hidden information, private key sk.
Output: Hidden information M.

Step 1: Move 4 bytes forward from the end of the last section and record the current pointer position SectionAddr.
Step 2: With SectionAddr as the starting address, read the contents of 4 cells as the address value Adrr, and determine whether the address is a JMP instruction for the unit where Adrr is located, or then the value of SectionAddr minus 1, continue to scan forward.
Step 3: If the jump instruction jumps exactly to the location where SectionAddr 4 is located, then the location of SectionAddr 4 is the post-migration function, and the starting address of the original function is in the two-word unit where SectionAddr is located, and then the transfer step 4, Otherwise, the value of Section Addr is reduced by 1, read 4 bytes in a row, and continue to scan forward.
Step 4: Read the starting address of the original function from the two-word unit where SectionAddr is located, skip the JMP instruction, read the length of the hidden information Len, and begin to extract hidden pieces of information that are len bytes in length.
Step 5: Determine whether the value of SectionAddr points to the beginning of the last section, and if so, the reverse scan ends, otherwise the value of SectionAddr is reduced by 4 and then transferred to step 2.
Step 6: The extracted pieces of M' secret information are reversed into secret information.
Step 7: Using the private key sk and asymmetric encryption algorithm RSA, the watermark information M' is decrypted. The decrypted information M = DeEncrypt(sk, M') is obtained (Where M is plaintext).

4 Experimental results and discussion

4.1 Results of the experiment

The PE files used in the experiment consist of three different types of files: some from the windows operating system's own applications, located in the windows system32 folder, such as notepad.exe, write.exe, winmine.exe, etc. Part of it is a common desktop application for users, such as qq.exe, thunder.exe, winRAR, 360sd.exe, and part of the application written for yourself. From which 200 PE files are randomly selected as test programs for experimentation.

In the experiment, the watermark information is embedded in all functions identifiable in each tester. The results of the experiment are as follows:

Table 2 Embedded capacity and bit rate of function migration methods

Filename	File length (bytes)	Number of identified functions	Function (average length)	Total hidden capacity (bytes)	Bit rate
write.exe	5,632	4	52.75	211	3.7464
notepad.exe	66,560	59	220.7966	13027	19.5718
telnet.exe	85,504	111	188.4505	20918	24.4644
winmine.exe	119,808	67	98.25373	6583	5.4946
qq.exe	99,744	40	142.525	5701	5.7156
360sd.exe	1,697,400	2805	193.3323	542297	31.9487
thunder.exe	1,808,176	4086	132.0952	539741	29.8500

As can be seen from Table 2, in general, the larger the file, the stronger the function, the more functions are recognized, the greater the hidden capacity.

5 Discussion

5.1 Covert analysis

The function migration method suggested in this paper moves the recognized function module to the last section to hide information in the original function code area. Using the services provided by the www.virscan.org website, the hidden PE file upload server will be hidden for virus scanning, the results show that the file is normal (the website provides up to 37 types of antivirus engines). Ability to resist the detection of common anti-virus software and the analysis of static reverse analysis tools.

5.2 Embedded capacity

The embedded capacity of a normal function migration method is related to the size of the PE file and the number of static library functions called in the file. In general, the larger the PE file, the more complex the function, the more static library functions are called, the more functions that can be identified and can be migrated, and the greater the embedded capacity.

5.3 Anti-filling attack experiment

Hiding information in the redundant space of the PE file, there are insufficient gaps in the hidden information that is too centralized, hidden location is easy to expose, hidden capacity is small, and the hidden information will be destroyed by filling the known redundant space with full 0 or full 1. Extending the last section of the PE file or adding a section to hide information, while solving the problem of hidden capacity, but as with the use of redundant space for information hiding, there is an over-concentration of information, hidden location disclosure problems, and because there is no integration with the program's main functional code, Using a full 0 or full 1 to fill forward from the end of the last section will break the hidden information, but the program will still function properly.

The function-based method is to hide the information in the original function code area by migrating the function code of the recognized system function or user-defined function to the last section of the PE file. Because the information is hidden in the code area of the original function module of the PE file, when using full 0 or full 1 to fill the attack forward from the end of the last section, the hidden information is not broken, while the fill attack will destroy the original function code that is migrated, resulting in the program not being able to run. Take notepad.exe, a notepad.exe that comes with the Windows operating system, for example, after using the function migration method to hide the information, the program can function properly and extract the hidden information.

The traditional method is to hide the secret information in the redundant space, data resource segment, and import table of PE file. There are some shortcomings, such as the known redundant space, the too concentrated hidden space, the easy destruction of hidden information, and the loose association between the hidden information and the key code of the program. Compared with the previous methods, the method proposed in this paper overcomes their shortcomings. Our method is to fuse the secret information with the instruction code of the program through function migration and store it in the code segment. The hidden information is scattered, and the adversary is difficult to determine the location of the secret information and instruction code, and the hidden information is coupled with the key code of the program. Once the secret information in the program is destroyed, the program will not be executed correctly. So, it is more secure and capable of resisting attacks than the previous methods.

6 Conclusion and future work

In this paper, a large-capacity information hiding algorithm based on function migration is presented. The PE file code section is disassembled through the disassembling engine processes functions recognition, and shifts the codes of recognized function. The design implements an algorithm that hides information by migrating the functional code of an identified static library function or user-defined function to the last section of the PE file. In this way, the hidden information is combined with the main functional code of the PE file, and the hidden information is coupled with the key code of the PE file, which further enhances the concealment and anti-attack of the system. The theoretical analysis and experimental results show that, compared with similar algorithms, the proposed algorithm integrates the information to be hidden with the program instruction code through function migration, and the algorithm hides the capacity and concealment, strong ability to resist attacks.

Abbreviations
PE: Portable executable; IAT: Import address table; RVA: Relative virtual address; DLL: Dynamic link library; MMX: Multimedia extensions; SSE: Streaming SIMD extensions

Acknowledgements
Thanks to the anonymous reviewers for their constructive suggestions to help improving this paper.

Authors' contributions
Our contributions in this paper were that the first author (Zuwei Tian) participated in the designing of the scheme and drafted the manuscript. The second author (Hengfu Yang) carried out code design, the experiments and participated in designing of the scheme. All authors read and approved the final manuscript.

Authors' information
Zuwei Tian received the B.E. degree in computer engineering from Xiangtan University, China, and the master's degree of computer science from National Defense Science and Technology University, China. He received the Ph.D. degree from Hunan University, China. He is a computer science professor of Hunan First Normal University, China. He leads a team of researchers and students in the areas of Information Security, such as information hiding. He has published more than 20 journals articles and his research has been funded by Natural Science Foundation Committee of China.
Hengfu Yang received the B.E. degree in computer engineering from Xiangtan University, China, and the master's degree of computer science from GuiZhou University, China. He received the Ph.D. degree from Hunan University, China. He is a computer science professor of Hunan First Normal University, China. His research interests include information hiding, image processing, and multimedia security.

Funding
This work is supported in part by the National Natural Science Foundation of China (61373132, 61872408), the Key Laboratory of informationization technology for basic education in Hunan province (2015TP1017), Hunan provincial higher education reform research project (2012[528]), Project of research study and innovative experiment for college students in Hunan Province(2017[873]).

Availability of data and materials
The datasets used and analyzed during the current study are available from the corresponding author on reasonable request.

Competing interests
The authors declare that they have no competing interests.

Received: 24 April 2020 Accepted: 11 November 2020
Published online: 06 January 2021

References

1. Z. Wu, S. Feng, J. Ma, Information hiding scheme and implementation of PE file. Comput. Eng. Appl. **41**(27), 148–150 (2005)
2. R. El-Khalil, A.D. Keromytis, *Hiding information in program binaries*, Proc of the 6th International Conference on Information and Communications Security (Springer, Berlin, 2004), pp. 287–291
3. R.K. Tiwari, G. Sahoo, A novel steganographic methodology for high capacity data hiding in executable files. Int. J. Internet Technol. Secured Trans. **3**(2), 210–222 (2011)
4. S.B. Che, S. Jin, G.W. Ling, in *International Conference on Computer Science and Education (ICCSE10)*. Software watermark research based on portable execute file (Hefei, 2010), pp. 1367–1372
5. Z. Sha, H. Jiang, A. Xuan, in *the 3rd International Conference on Genetic and Evolutionary Computing (WGEC09)*. Software watermarking algorithm by coefficients of equation (Guilin, 2009), pp. 410–413
6. X. Wang, Y. Wang, X. Zhang, et al., Research on PE file software watermark against similarity attack. Netw. Secur. Technol. Appl., 82–84 (2007)
7. A.A. Zaidan, B.B. Zaidan, A.W. Naji, et al., in *International Conference on Advanced Management Science (ICAMS09)*. Approved undetectable-antivirus steganography for multimedia information in PE-file (Singapore, 2009), pp. 437–441
8. H. Alanazi, H.A. Jalab, A.A. Zaidan, et al., New framework of hidden data with in non multimedia file. Int. J. Comput. Netw. Secur. **1**, 46–53 (2010)
9. A.W. Naji, A.A. Zaidan, B.B. Zaidan, Challenges of hidden data in the unused area two within executable files. J. Comput. Sci. **1**, 890–896 (2009)
10. A.A. Zaidan, B.B. Zaidan, A.W. Naji, et al., in *International Conference on Information management and engineering (ICIME09)*. Securing cover-file of hidden data using statistical technique and AES encryption algorithm (Malaysia, 2009), pp. 35–40
11. A. Haveliya, A new approach for secret concealing in executable file. Int. J. Eng. Res. Appl. **2**(2), 1672–1674 (2012)
12. B.B. Zaidan, A.A. Zaidan, F. Othman, et al., in *Proceeding of the International Conference on Cryptography, Coding and Information Security*. Novel approach of hidden data in the unused area 1 within exe files using computation between cryptography and steganography (Paris, 2009), pp. 1–22
13. M.R. Islam, A.W. Naji, A.A. Zaidan, et al., New system for secure cover file of hidden data in the image page within executable file using statistical steganography techniques. Int. J. Comput. Sci. Inf. Secur. **7**(1), 273–279 (2009)
14. B.B. Zaidan, A.A. Zaidan, F. Othman, New technique of hidden data in PE-file with in unused area one. Int. J. Comput. Electrical Eng. (IJCEE) **1**(5), 669–678 (2009)
15. A.W. Naji, A.A. Zaidan, B.B. Zaidan, et al., New approach of hidden data in the portable executable file without change the size of carrier file using distortion techniques. Int. J. Comput. Sci. Netw. Secur. **9**(7), 218–224 (2009)
16. A.A. Zaidan, B.B. Zaidan, A.J. Hamid, A new system for hiding data within (unused area two + image page) of portable executable file using statistical technique and advance encryption standared. Int. J. Comput. Theory Eng. **10**(5), 125–131 (2010)
17. D. Shin, Y. Kim, K. Byun, et al., in *Proceedings of the 6th Australian Digital Forensics Conference*. Data hiding in windows executable files (Perth, 2008), pp. 1–8
18. L. Qian, F. Yong, D. Tan, Z. Changshan, Research on information hiding technology based on unlimited capacity of PE file. Comput. Appl. Res. **28**(7), 2758–2760 (2011)
19. W. Wei, K. Liu, X. Wan, High capacity information hiding based on PE file format. J. Nanjing Univ. Sci. Technol. **39**(01), 45–49 (2015)
20. Y. Li, X. Shi, Research on PE file information hiding technology. Netw. Secur. Technol. Appl. (11), 51–52 (2017)
21. X. Xu, X. Xu, H. Liang, et al., Information hiding research and scheme implementation of PE file resource section. Comput. Appl. **27**(3), 621–623 (2007)
22. D. Qingfeng, Y. Wang, Z. Kaize, W. Xi, Information hiding scheme based on PE file resource data. Comput. Eng. **35**(13), 128–130 (2009)
23. Z. Tian, Y. Li, L. Yang, Research on PE file information hiding technology based on import table migration. Comput. Sci. **43**(01), 207–210 (2016)
24. J. Xu, J.F. Li, Y.L. Ye, et al., An information hiding algorithm based on bitmap resource of portable executable file. J. Electron. Sci. Technol., 181–184 (2012)
25. D. Qingfeng, W. Yanbo, Z. Xiongwei, Z. Kaize, Spread spectrum software watermarking scheme based on the number of import function references. Comput. Res. Dev. **46**(supply), 88–92 (2009)
26. F. Long, J. Liu, X. Yuan, A software watermark for transforming the structure of PE file import table. Comput. Appl. **30**(1), 217–219 (2010)
27. A.P. Namanya, I.U. Awan, J.P. Disso, M. Younas, *Similarity hash based scoring of portable executable files for efficient malware detection in IoT. Future Generation Computer Systems* (2019)
28. S.L. Shiva Darshan, C.D. Jaidhar, Performance evaluation of filter-based feature selection techniques in classifying portable executable files. Proc. Comput. Sci. **2018**, 125 (2018)
29. X. Wang, J. Jianming, Z. Shujing, B. Liang, A fair blind ignature scheme to revoke malicious vehicles in VANETs, computers. Mater. Continua **58**(1), 249–262 (2019)
30. J. Wang, H. Wang, J. Li, X. Luo, Y.-Q. Shi, S. Kr, Jha, Detecting double JPEG compressed color images with the same quantization matrix in spherical coordinates. IEEE Trans. CSVT (2019). https://doi.org/10.1109/TCSVT
31. J. Wang, T. Li, X. Luo, Y.-Q. Shi, S. Jha, Identifying computer generated images based on quaternion central moments in color quaternion wavelet domain. IEEE Trans. CSVT **29**(9), 2775–2785 (2018)
32. K. Chen, Z. Liu, Current situation and progress on decompilation research. Comput. Sci. **28**(5), 113–115 (2001)

Publisher's Note