SOFTWARE MANIPULATIVE TECHNIQUES OF PROTECTION AND DETECTION: A REVIEW

M. A. Ibrahim[1,2], Z. Shukur[2], N. Zainal[3] and Abdo A. A. Al-Wosabi[2]
[1]National Metrology Institute of Malaysia, Selangor, Malaysia
[2]Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Selangor, Malaysia
[3]Faculty of Engineering and Built Environment, Universiti Kebangsaan Malaysia, Selangor, Malaysia
E-Mail: mdazwan@sirim.my

ABSTRACT

Over the last decade, many studies have been conducted concerning the protection of software. Software piracy, tampering and stealing became the major concern of various parties such as software developers, suppliers, traders and consumers. This paper summarizes some of the related methods in software security such as steganography, obfuscation and cryptography. Also some of the most applicable techniques in securing software from manipulation such as software watermarking, fingerprinting and software birthmarking are reviewed in this paper.

Keywords: steganography, obfuscation, fingerprint, watermark, birthmark, software security, software protection, tamper detection, software tampering.

INTRODUCTION

In the digital era, everything is now relies on software. Whether in banking, trades, medical, production, entertainment and education, software plays an important part. Weaknesses in software related to security bring a lot more chaos than we could ever expect. Software vulnerable leads to software piracies, code stealing and software tampering. This does not only affecting the software industries, but brings much more troubles such as in economic and legal situation, where people nowadays tend to tamper or manipulate software in the favors of their purposes in every sectors.

Illegal manipulation of software is one of the biggest issues in software security. There have been many extensive studies related to the software security such as steganography, obfuscation, watermarking, birthmarking and more. Some of them have existed in literatures from the studies done years ago, but are still being practiced until today.

Discussing the matter of security, there is no exact definition to measure the security and robustness. Both of the terms are relatively measures. However, a system can be considered to be robust and secure if the cost of breaking into the system is higher than the cost of the system and the time required to break the system exceeds the lifetime of the data [1].

Moreover, the difficulty to determine or prove that an illegal copy (software tampering) is being distributed and prosecute the violators during court case trial is one of the main problems need to be solved.

There are a number of real life cases where tampering could be a serious threat to community, for instance; a case as of petrol station in Silibin, Ipoh has been reported in year 2013 by the Malaysian enforcement authority where the owner had manipulated their fuel pumps to gain more profit. Similar cases also have been reported in India in year 2008 [2].

There are several studies and techniques that applicable for curing such problems. This paper summarizes and focuses on the techniques that applicable from the previous studies that haves been conducted and it is organized as follows. The first section of the paper is the introduction part which discussed the software manipulation issues, second is the methodology section, which explain the methods used in organization of this review, and third is the related method in supporting software securing. Software watermarking, fingerprinting and software birthmark come next in section four.

METHODOLOGY

This section discusses the methodology used in preparing this paper from starting from data collection, search terms and searchable database.

Data collection

A literature search has been conducted in 2015, for the purpose of compiling and reviewing studies that have been conducted related with the title of this paper. The search also cover patents that has been filed related with the title of interests. The time period covered by this paper was from year 1994 to 2015.

Search term

Search terms used were based on the software security synonymous keywords such as “tamper detection”, “software watermark”, “software fingerprint”, “software birthmark”, “software steganography”, “obfuscation”, “tamper prevention”, “portable executable watermark”, “PE watermark”, “PE specification”, “watermark patent”, “java watermark”, “java birthmark”, “binary executable protection”, ”binary watermarking”, “text fingerprint”, “document fingerprint”, “digital fingerprint”, “software copy detection”, “software theft”, “code theft”, “code tampering detection”, “cryptography”, “steganography”. Search outcome were then filtered by relevancy of the title of this paper.

VOL. 10, NO. 23, DECEMBER 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences
©2006-2015 Asian Research Publishing Network (ARPN). All rights reserved.
www.arpnjournals.com

Searchable database

The database searched for the study were online specific for literature databases such as Mendeley Literature Search, Google Scholars, IEEE Xplore Digital Library, ACM Digital Library, Scopus, Springer and Science Direct. Google Patent was specifically used in the search for patent that are related with this paper.

RELATED METHODS

The following are fundamental methods that support the techniques in securing software. Most of the methods in this section are common in information security such as steganography, cryptography and obfuscation. Selection of papers is made so that only the methods related with securing software are presented.

Steganography

Steganography is defined as the method of hiding information by using innocuous carriers by means of covering the existence of the secret information. The word steganography itself was derived from ancient Greek, which means to cover or hide. It is not intended to replace cryptography but rather to complement it. By concealing information with encryption, it will reduce the chances of the information being revealed [3].
There are three main elements in a steganography system (a) Cover message; (b) Embedded message and (c) Steganographic key. Figure-1 depicts a typical steganography system [4].

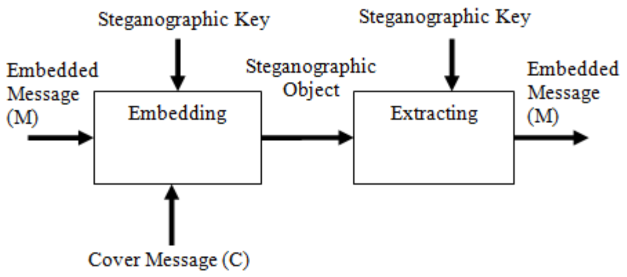

Figure-1. Typical steganography system.

Cover message (C) is the carrier message where the purpose is to bring the embedded message along with it. The cover message is to be combined with embedded message (M) using a specific steganographic algorithm. The cover message itself may not have any connotation with the embedded message.

Embedded message is the actual message to be sent which is to be embedded within the cover message. It is also meant to be hidden and unseen by normal reader unless reversed by a specific steganographic algorithm. The combination of the cover message and embedded message is called steganographic message. The cover message may not be of the same data types with the embedded message, but the cover message and the steganographic message must have same data type.

Steganographic key is the additional information or data, which to be used in the process of embedding and extracting. Steganographic key is an optional element of which the sender may use or not. Adding steganographic key in the element makes the embedded message even more difficult to reveal.

There are five main criteria to be considered when designing a steganography system. (a) Capacity, the number of bits hidden and to be recovered successfully by the steganography system; (b) Robustness, the ability of the information to remain intact after the cover message has undergone process of changes and transformations; (c) Undetectable, the format and the physical structure of the cover message remain the same after the process of insertion; (d) Invisibility Perceptual Transparency, does not make any detectable changes by human to the cover message after the process of insertion of hidden information by exploiting human perceptual capabilities and (e) Security, the embedded message is difficult to be removed after being discovered a third party.
The importance of above criteria depends upon the application and environment of the steganography system [5].
A quite recent example of study in steganography that works at the firmware level is as explained by [6]. They introduced a method of concealing data in a hard disk drive by manipulating the drive's defect control systems firmware. The defect control system is part of the firmware functions, which transparent to the users and works 'beneath' the control of operating systems. Any defects on the drive will be recorded within the P-list and G-list of the firmware. Listed sectors will be bypassed by the drive's electronics and data will on that particular sectors will be inaccessible.

Cryptography

Cryptography is the method of scrambling data into something that is not understandable. Normally steganography and cryptography are used together as both of them complementing each other. Often, before a message is to be hidden using steganography technique as discussed above, it will be first encrypted so that even if the hidden information is successfully revealed; it would still be very difficult for the unintended party to understand the actual meaning of the message.

The original message to be sent is called plaintext, which will undergo the process of encryption and transforming it into a new form of message which is called cipher text. The intended recipient then needs to decrypt the cipher text using a key provided by the sender in order to recover the original message.

There are two ways of processing the plaintext: stream cipher and block cipher. In stream cipher mode, the plaintext is coded 'on-the-fly' with a continuous code seeded by a generator [7]. Whilst in the block cipher mode, the data is processed in a predefined blocks size. The plaintext is divided into groups of predefined block size before they are to be encrypted.

There are two types of encryption schemes, asymmetric and symmetric encryptions. Asymmetric encryption uses a pair of key to encrypt and decrypt while symmetric encryption uses only single (shared) key for both operations.

There are two types of keys in asymmetric encryption. First is public key cryptography, which the key is known by the public and second is private key cryptography which, the key is own only by the user. Let say Alice would like to send private message to Bob, Alice and Bob needs to negotiate the encryption method to be used in the first place. After the agreement, Bob sends to Alice a public key for Alice to use in encrypting her message. After the message arrived, Bob uses his private key to decrypt and retrieve Alice's original message.

There are few major asymmetric encryption algorithms such as RSA (Rivest, Shamir and Adleman), ECC (Elliptic Curve Cryptography) and El Gamal. The RSA algorithm was introduced since 1977 and it is the most widely used asymmetric algorithm today. It provides good level of security but it is slow in encrypting speed [8]. Next, Victor Miller of IBM, and Neal Koblitz of the University of Washington introduced ECC in 1985. ECC has the advantage of smaller footprints where it requires smaller processing power due to smaller key size while providing same level of security compared to RSA [9].

Taher ElGamal first described the ElGamal encryption in 1984. A studied suggested that the ElGamal algorithm is more secured compared to RSA algorithm but at the same time it is much slower in processing [10].

The strongest symmetric encryption algorithm to date is the well-known AES standard based on Rijndael algorithm [11]. AES stands for Advanced Encryption Standard which is defined under Federal Information Processing Standard (FIPS) 192 [12]. AES supersedes DES (Data Encryption Standard) after DES has been successfully broken by attackers [13].

AES operates on an array of bytes of 4×4 (128 bits) which is called a state. It supports key length of 128, 192 and 256 bits. The encryption and decryption is done via four transformations for a specific number of rounds (10, 12 or 14) based on the key length stated above [14].

There are many variants of AES encryption introduced by cryptography scholars to date. Each of the variant basically optimizes AES to meet for specific purpose such as HD multimedia encryption, reducing processing load for low performance devices, wireless transmission and others. Variant of AES is known as MAES (Modified Advance Encryption Standard).

Obfuscation

Obfuscation is a technique to complicate the control flow of an instruction stream and data structures which contains sensitive information so that to mitigate from code reverse engineering [15].

A study in [16] defines that the transformation of original program P into an obfuscated program Ṕ (as a transformation of Ʈ) shall have the same observable behavior of both. P and Ṕ shall be the same in terms of their functionalities to the user. It is aside of the performance losses due to the obfuscation transformation.

Virus writers often employ obfuscation-like techniques to prevent it from being detected along with tamper-proof alike technique to prevent it from being removed easily [16].

There has been a study of an implementation of obfuscation at assembly level by inserting junk bytes [17]. Junk bytes are used to fool disassembler at selected instruction stream and it serve for two main purposes, it must be partial instructions; and it shall not interfere with existing instructions such that the partial instructions must be unreachable during runtime.
A hybrid combination of obfuscation between static and dynamic analysis has been discussed in [18]. Static analysis obfuscation is implemented by adding branching functions with statically obfuscation algorithms to control the flow of software, resulting large number of small chunks of code blocks. The control flow of the instructions and the information on how to connect between small chunks of codes to form a sequence of valid program are unknown. On the dynamic obfuscation sides, the control flow graph of the software is diversified so that it contains much more control flow paths in order to make debugging difficult. The small chunks of codes is diversified in terms of the flow (i.e. semantically identical but syntactically different) and adding input dependent branches so that different chunks of codes get executed with different inputs.

Some of the fundamental solutions in the techniques of securing software have been discussed. In the next section, we will be discussing the major techniques to be used in securing software such as software watermark, fingerprinting and software birthmark.

SOFTWARE WATERMARK

Software watermarking can be defined as the process of embedding additional information into software, without interrupting the functionality of the software itself.

The earliest patents were filed in 1994, based on the concepts of software watermarking. The watermarking proposed by [19] are methods for, identifying unauthorized copies and; to provide a method for identifying the source of unauthorized copies. In another patent in the same year [20], an assignee claiming a method and apparatus for serializing and validating copies of software, and thus, possibilities of disabling the functionalities of the software whenever an unauthorized copy is found.

Two years later, a patent had been filed by Microsoft Corporation in 1996 [21], utilizing software watermarking concepts. The method is by rearranging blocks of codes so that arrangement of code blocks become as its unique identification on each software distribution.
Ever since the popularity of digital watermarking started to increase, the terms "watermarking" became more ambiguous due to the many kinds of watermarking-alike technology appearing. A taxonomy in software

watermarking has been specified in a study [22], in order to clarify the sub-terms emerged in the watermarking technology.

Some studies stated that, a strong watermark shall be able to withstand these four types of attacks [1, 23] (a) Subtractive Attack, attempt of removing watermark completely so that there will be no sign of watermark; (b) Distortion Attack, attempt of damaging or distorting the watermark so it will be unrecognizable; (e) Additive Attack, action of overwriting the original watermark with a new one by adversary and (f) Collusive Attack, locating the watermark by differentiating between different fingerprints of watermark.

An attack is considered to be successful if the watermark is failed to be extracted from the attacked software, and also if the de-watermarked performance of attacked program is considered useful by the attacker [23]. There are two major types of software watermarking modes of operations; known as dynamic and static watermarking. Static watermarks are embedded into the code and/or data of a program while the dynamic watermark is applied during the program's execution state [16, 24, 25].

There is another method of watermarking called graph watermarking. This type of watermarking is available in both static and dynamic type of watermarking. The first graph based static software watermarking was first introduced in [26]. The idea is to convert both software and watermark codes into digraphs and adding new edges between them by adding additional function calls between the software and watermark codes. The complement of the dynamic sides was firstly introduced in [27], which is a type of dynamic graph based software watermarking, called CT algorithm. It is implemented by embedding the watermark in the topology of dynamic heap data structures, generated by the code algorithm, while a secret key of series sequence is fed into the system, the watermark (graph structure) will be appeared and can be identified by the recognizer.

Spread-spectrum method of watermarking was proposed in [28], by analyzing functions representing consecutive of assembly instructions as a type of statistical object. The frequency counts of set of consecutive instructions become the marking object. Another spread-spectrum based is introduced [29], utilizing multimedia concept of watermarking in software, where vector r is extracted from the properties of the running programs. The call graph is altered using algorithm to embed the watermark.

There are other tremendous researches in Java based watermarking [25], such as a method proposed using dummy Java class file and a dummy call instruction to be inserted into a Java program. The dummy conditional instruction is written such that the return value for the condition will never make the method within the class to be executed. The dummy Java class contains the watermark information. This method has been proposed to be used with obfuscation techniques in order to prevent the attacker from locating the dummy methods [30].

Besides that, there are two types of watermarking robustness, each with their own unique application. First is robust watermarking; which is the ability of the watermark to hold themselves and can be successfully extracted even after the file suffered from maliciously attacked, and secondly the fragile watermarking of which the mark will be destroyed upon any tampering circumstances. The robust watermarking is used in as copy protection and proof of ownership while the fragile watermarking is mainly used in tampering detection and as a proof of integrity.

Zero-watermark is another concept of watermarking [31–33]. The technique is quite different from other watermarking schemes since there is no additional watermark message to be embedded into the original message, but rather utilizes the combination of birthmark and watermark approach by calculating registration message along with the birthmark data. The technique can be seen in Feature n-gram Set (FnGS) concept, which method has been proposed in [34].

The terms Zero-watermark above shall not be confused with Zero-knowledge watermarking [35–37]. In zero-knowledge watermarking, the existence of watermark can be soundly proven without revealing the details of the watermark itself. This prevents sensitive data to be disclosed to unnecessary party.

Microsoft has released specifications structure of executable file. This reference shall be used to embed watermarks on portable executable. Figure-2 shows the typical layout of an executable [38].

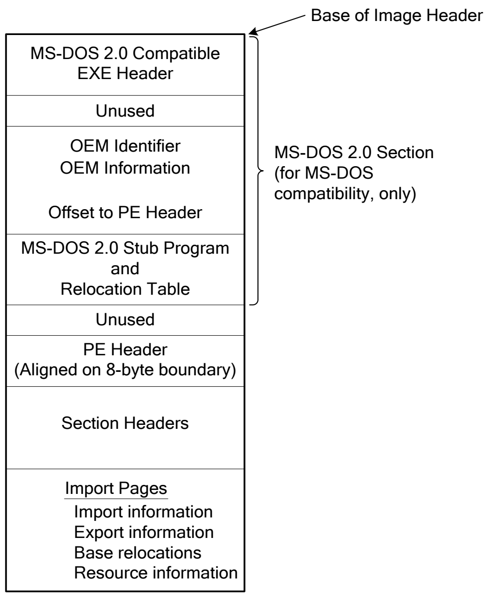

Figure-2. Portable executable file layout.

Recently, there have been active studies conducted on watermarking of portable executable (PE) file, a watermark in binary form of executable. A watermark is inserted by analyzing suitable space for inserting watermark using standard profiling of PE file structure. This implementation requires high level of understanding of the format of PE specifications and Relative-virtual Address (RVA) formatting. RVA is a position unit within the PE and is used as a start-address of a PE file loaded into memory [39].

There are several studies on watermarking on various locations of PE layout as; in unused area number 1 [40], Unused area number 2 [4, 41], Combination of Unused Area number 1 and Unused Area number 2 [42], Import Pages [43] and Bitmap Resources area [44]. In another study, watermarking data within triplex space for EXE file has been presented. By making use of three possible spaces which are Unused Area 1, Unused Area 2 and Image Pages areas. This method could hold around 28% of overall size of cover file before watermarking [45].

In a study in [39], two new techniques of embedding hidden information into the PE have been introduced: (a) Utilizing slack space, where slack spaces are unused spaces within each section. The size of data to be inserted is limited, depending on the size of slack spaces; (b) At the .text section, inserting data into this area does make the overall size of PE changed. Blindly inserting will make the PE not executable. Modifications are required on both section table and IMAGE_OPTIONAL_HEADER to make the PE usable again. The advantage being there is no limit on the hidden information to be inserted and the execution results are not affected.

A modification at the code level approach has been introduced [46]. The basic idea is by inserting redundant functions within the executable file and they are distributed randomly within the executable file, thus making it difficult to be traced. This method theoretically has unlimited watermark data size capacity.

Currently, most of the watermarking scheme proposals are to protect the ownership of the piece of media/software, but there is an urgent need of watermarking as integrity verification of software in practical applications [47].

A good example of software watermarking that has been used outside its ordinary usage as introduced in [48]. The author proposed a new virus detection mechanism via software watermarking. This is a type of watermarking which is implemented within the PE binary format. The concept is by embedding a fragile watermark along with the identity of the PE carriers. This method has been tested and proven to have high detection rates for known and unknown viruses compared to conventional virus signature-based detection methods.

We can see a lot of potential in software watermarking technique, where it has been used in various situations as discussed. In the next section we will be discussing on another near-similar concept of watermarking which is called fingerprinting.

FINGERPRINTING

Fingerprinting is basically the same as watermarking, except that fingerprinting embeds unique identifier information on each distribution copies of software. This may not only detect an occurrence of software violation copies, but also able to trace the violator. A fingerprint may include vendor, product or customer information [16].

A quick substring matching algorithm has been proposed in [49]. This method is the earliest version of fingerprinting based on k-grams concept. Winnowing is introduced later in [50], an algorithm for document fingerprinting, based on the k-grams method as well.
Fingerprinting Java based program using Java class file obfuscation is presented in [51], based on watermarking scheme introduced in [30], with addition of some modifications to make it as a fingerprinting approach.

A Patent has been filed by [52] on fingerprinting. The implementation is by inserting NOP (No Operation) codes into an executable in a pattern as identification of the fingerprint. NOP code is a type of instruction, which does nothing within the program functions. In this case, it is inserted purposely with pre-determined pattern for identification purposes.

Another alternative method in securing software called software birthmark will be discussed in the next following section.

SOFTWARE BIRTHMARK

One of the less popular methods on securing a copy of software is called software birthmark. It has quite a different approach compared to software watermark. The general concept of software birthmark is the same that is found in the computer virus signature concepts; to produce a unique identification of software.
There are two important characteristics that differ between the software watermark and software birthmark [53]; (a) In software watermark, it is often necessary to embed external information or data or code within carrier software, whereas it is not required in software birthmark; (b) Birthmark could not be used to identify ownership, or source of distribution but rather to confirm that software or code whether it is in partially or in fully, is a reproduction of others.

In software birthmark, the inherent characteristic from the software itself is to be extracted, whilst for software watermark, the pre-embedded information is to be extracted. Software watermark provides further some evidence on the ownership information, but the application of software birthmark is strictly and serves better in proving evidence of software/code theft.

A software birthmark is a method of producing unique characteristic on each of the software or code. There will be no same markings on each of the software. If same marks ever found on two or more different copies of software, it proves that a copy violation has occurred.
If only part of the codes is stolen, and integrated into new software, which is then distributed. It makes even more difficult to detect or prove there is a code theft [54].
A strong birthmark shall be able to withstand from attacks via code transformation has been applied in order to hide the theft [53].

Same as in software watermarking, birthmark generally can be divided into two categories; static analysis and dynamic analysis. Static birthmark is derived from the syntactical instruction code structure of the software while the dynamic birthmark is implemented during the runtime [49].

It can be further divided into few more sub-categories [55] as (a) Static source code based birthmark [54]; (b) Static executable code based birthmark [56]; (c) Dynamic WPP based birthmark [53]; (d) Dynamic API based birthmark [57, 58] and (e) Dynamic behavior based birthmark [59].

Based on static analysis birthmark, [60] proposed four new static based analysis of Java class file code theft detections; Constant Values in Field Variables (CVFV), Sequence of Method Calls (SMC), Inheritance Structure (IS) and Used Classes (UC). These four methods were further evaluated by [54] on the performance with practical applications and tolerance against program transformation.

A strong static executable code based birthmark has been proposed in [56]. The method is implemented by tabulating the sequence of instruction and applying k-gram concepts for analyzing its opcode. The method of k-gram is previously used for document fingerprinting and it is strong in plagiarism detection.

A dynamic birthmark based called Whole Program Paths (WPP) is introduced in [53]. WPP is originally used to represent the dynamic control flow of a program. This method is weak against some many of the semantic-preserving transformation such as loop unwinding and in-lining functions. Within these limitations, WPP is not suitable in environments where various compilers are available such as Windows [55, 58].

In another study, Tamada proposed new approaches based on dynamic API (Application Program Interface) birthmarking called EXESEQ (Sequence of API Function Calls Birthmark) and EXEFREQ (Frequency of API Function Calls Birthmark) [54, 58]. Since the API call is being analyzed, this method is suitable within windows environment. This is based on the facts that it is difficult to modify API calls within a binary code without interrupting its operational behavior of executions. Therefore, API call is used as a signature of a program.

Based on System Dependent Graph Based (SDCG), a new technique is proposed [59] called System Call Short Sequence Birthmark. Short sequence of system calls has been used previously to detect abnormality of behavior of a program. In SDCG, program's behavior is represented by using graph. Via these two techniques, a new method has been developed to detect behavior similarities between programs.

As discussed above, software birthmark has its own potential with different kind of approaches and serves better in proving copy violations. The main objective of studies in this area is mainly to find the best approach to determine a unique identifier on each software within its very original form.

DISCUSSIONS

There are three major techniques that have been discussed; software watermarking, fingerprinting and birthmarking. Table-1 briefly compares each of the technique.

Table-1. Comparison summary between different techniques.

Technique	Changes to original content	Concept	Usage
Software Watermark	Yes. Except zero watermark	Embeds additional information. Could be either visible or invisible	Copyright protection. Proof of ownership. Tampering detection. Integrity checker.
Fingerprinting	Yes	Embeds unique identification to each distribution copies.	Tracing violator. Identifying source of distributor.
Software Birthmark	No	Generate unique identifier to each for each software / code.	Proof of ownership. To determine whether a software or code either in fully or partially, is a reproduction of others

All these techniques in combination with a number of fundamental security related solutions such as cryptography, steganography and obfuscation, could possibly achieve a good protection scheme which can be used to protect against software stealing, piracy, tampering and also malware as well as virus detection and curing.

CONCLUSIONS

In this paper, we reviewed some of the techniques that applicable to securing software from manipulation from the definitive to the recent years of study. It is suggested that there are different kind of methods to securing software depending on the suitability of the application and environment.

Our aim from this review is to propose an effective method and framework in the future for detecting and securing software by utilizing some of the methods discussed above. The application would be on the protection of software based instruments that relates with trade and consumer. This would directly benefit the both business and consumers community in by means of trustworthy transaction.

REFERENCES

[1] Q. Albluwi and I. Kamel. 2006. Watermarking Essential Data Structures for Copyright Protection. In: 5th International Conference, CANS 2006, Suzhou, China, December 8-10. Proceedings, 2006, pp. 241–258.

[2] Abdo Ali Abdullah Al-wosabi, Z. Shukur, and M. A. Ibrahim. 2015 Framework for Software Tampering Detection in Embedded System. In: 5th International Conference on Electrical Engineering and Informatics.

[3] Z. K. Al-ani, A. A. Zaidan, B. B. Zaidan, and H. O. Alanazi. 2010. Overview : Main Fundamentals for Steganography. J. Comput., Vol. 2, No. 3, pp. 158–165.

[4] A. W. Naji, A. A. Zaidan, and B. B. Zaidan. 2009. Challenges of Hidden Data in the Unused Area Two within Executable Files. J. Comput. Sci., Vol. 5, No. 11, pp. 890–897.

[5] Y. Al-Nabhani, A. A. Zaidan, B. B. Zaidan, H. A. Jalab, and H. O. Alanazi. 2010. A new system for hidden data within header space for EXE-file using object oriented technique. In: Proceedings - 2010 3rd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2010, 2010, Vol. 7, pp. 9–13.

[6] I. Sutherland, G. Davies, and A. Blyth. 2011. Malware and steganography in hard disk firmware. J. Comput. Virol., Vol. 7, No. 3, pp. 215–219.

[7] K. Saranya, R. Mohanapriya, and J. Udhayan. 2014. A Review on Symmetric Key Encryption Techniques in Cryptography. Int. J. Sci. Eng. Technol., Vol. 3, No. 3, pp. 539–544.

[8] R. Bhanot and R. Hans. 2015. A Review and Comparative Analysis of Various Encryption Algorithms. Int. J. Secur. Its Appl., Vol. 9, No. 4, pp. 289–306.

[9] K. Gupta, S. Silakari, N. Koblitz, and V. Miller. 2011. ECC over RSA for Asymmetric Encryption : A Review unexpected application of elliptic curves in integer. Int. J. Comput. Sci. Issues, Vol. 8, No. 3, pp. 370–375.

[10] A. Sharma, J. Attri, A. Devi, and P. Sharma. 2014. Implementation & Analysis of RSA and ElGamal Algorithm. In: National Conference on Advances in Basic & Applied Sciences, 2014, Vol. 2, No. 3, pp. 125–129.

[11] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback. 2000. Report on the Development of the Advanced Encryption Standard (AES).

[12] M. Pitchaiah, P. Daniel, and Praveen. 2012. Implementation of Advanced Encryption Standard Algorithm. Int. J. Sci. Eng. Res., Vol. 3, No. 3, pp. 1–6.

[13] S. M. Wadi and N. Zainal. 2013. A Low Cost Implementation of Modified Advanced Encryption Standard Algorithm Using 8085A Microprocessor. J. Eng. Sci. Technol., Vol. 8, No. 4, pp. 406–415.

[14] A. Abdulgader, M. Ismail, N. Zainal, and T. Idbeaa. 2015. Enhancement of AES Algorithm Based on Chaotic Maps and Shift Operation for Image Encryption. J. Theor. Appl. Inf. Technoogy, Vol. 71, No. 1.

[15] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. 1997. In: Technical Report 148 Department of Computer Science University of Auckland July, 1997, No. 148, p. 36.

[16] C. S. Collberg and C. Thomborson. 2002. Watermarking, tamper-proofing, and obfuscation - Tools for software protection. IEEE Trans. Softw. Eng., Vol. 28, pp. 735–746.

[17] C. Linn and S. Debray. 2003. Obfuscation of executable code to improve resistance to static disassembly. In: Proceedings of the 10th ACM conference on Computer and communications security.

[18] S. Schrittwieser and S. Katzenbeisser. 2011. Code obfuscation against static and dynamic reverse engineering, In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2011, vol. 6958 LNCS, pp. 270–284.

[19] K. Holmes. 1994. Computer software protection. US5287407 A.

[20] P. R. Samson. 1994. Apparatus and method for serializing and validating copies of computer software. US 5287408 A.

[21] R. I. Davidson and N. Myhrvold. 1996. Method and system for generating and auditing a signature for a computer program. US 5559884 A.

[22] J. Nagra, C. Thomborson, and C. Collberg, A functional taxonomy for software watermarking. 2002. In: Aust. Comput. Sci. Commun., 2002, Vol. 24, No. 1, pp. 177–186.

[23] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, and M. Stepp. 2004. Dynamic path-based software watermarking. in ACM SIGPLAN Notices, 2004, Vol. 39, No. 6, p. 107.

[24] J. Hamilton and S. Danicic. 2011. A survey of static software watermarking. In: 2011 World Congress on

Internet Security (WorldCIS-2011), 2011, pp. 100–107.

[25] C. Collberg, G. Myles, and A. Huntwork. 2003. Sandmark - A tool for software protection research. IEEE Secur. Priv. Arch. Vol. 1 Issue 4, July 2003, Vol. 1, No. 4, pp. 40–49.

[26] R. Venkatesan, V. Vazirani, and S. Sinha. A Graph Theoretic Approach to Software. In: 4th International Information Hiding Workshop, 2001, pp. 157–168.

[27] C. Collberg and C. Thomborson. 1999. Software watermarking: Models and Dynamic Embeddings. In: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '99, 1999, pp. 311–324.

[28] J. Stern, G. Hachez, F. Koeune, and J.-J. Quisquater. 2000. Robust Object Watermarking: Application to Code. In: Third International Workshop, IH'99, 2000, Vol. 1768, pp. 368–378.

[29] D. Curran, N. J. Hurley, and M. O. Cinneide. 2003. Securing Java through Software Watermarking. in PPPJ '03 Proceedings of the 2nd international conference on Principles and practice of programming in Java, 2003, pp. 145–148.

[30] A. Monden, H. Iida, K. Matsumoto, K. Inoue, and K. Torii. 2000. A practical method for watermarking Java programs. In: Proceedings 24th Annual International Computer Software and Applications Conference. COMPSAC2000.

[31] Z. Jalil, A. M. Mirza, and M. Sabir. 2010. Content based Zero-Watermarking Algorithm for Authentication of Text Documents. Int. J. Comput. Sci. Inf. Secur., Vol. 7, No. 2, pp. 212–217.

[32] Z. Jalil, A. M. Mirza, and T. Iqbal. 2010. A Zero-Watermarking Algorithm for Text Documents Based on Structural Components. 2010 Int. Conf. Inf. Emerg. Technol. ICIET 2010, pp. 1–5.

[33] H. Ishizuka, I. Echizen, K. Iwamura, and K. Sakurai. 2014. A Zero-Watermarking-Like Steganography and Potential Applications. In: 2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 2014, pp. 459–462.

[34] B. Lu, F. Liu, X. Ge, and P. Wang. 2008. Feature n-gram Set Based Software Zero-Watermarking. In 2008 International Symposiums on Information Processing, 2008, pp. 607–611.

[35] A. Adelsback and A.-R. Sadeghi. 2005. Zero-Knowledge Watermark Detection and Proof of Ownership. In: 2004 Information Hiding Workshop, Proceedings, LNCS 3200, 2005, Vol. 3200, pp. 239–252.

[36] S. Bhattacharya and a. Cortesi. 2010. Zero-knowledge Software Watermarking for C Programs. In 2010 International Conference on Advances in Communication, Network, and Computing.

[37] Z. Fu, X. Sun, J. Shu, and L. Zhou. 2014. Plain Text Zero Knowledge Watermarking Detection Based on Asymmetric Encryption. In: International Conferences, ISA, CIA, 2014, Vol. 48, No. Cia, pp. 126–134.

[38] Microsoft. 2013. Microsoft PE and COFF Specification. Microsoft Corporation, US.

[39] D. Shin, Y. Kim, K. Byun, and S. Lee. 2008. Data Hiding in Windows Executable Files. In: Proceedings of the 6th Australian Digital Forensics Conference.

[40] A. A. Zaidan, B. B. Zaidan, and F. Othman. 2009. New Technique of Hidden Data in PE-File with in Unused Area One. Int. J. Comput. Electr. Eng., Vol. 1, No. 5, pp. 642–650.

[41] A. W. Naji, A. A. Zaidan, B. B. Zaidan, S. A. Hameed, and O. O. Khalifa. 2009. Novel Approach for Secure Cover File of Hidden Data in the Unused Area within EXE File Using Computation between Cryptography and Steganography. J. Comput. Sci., Vol. 9, No. 5, pp. 294–300.

[42] W. F. Al-khateeb and S. A. Hameed. 2009. New Approach of Hidden Data in the portable Executable File without Change the Size of Carrier File Using Statistical Technique. Int. J. Comput. Sci. Netw. Secur., Vol. 9, No. 7, pp. 218–224.

[43] M. H. Jawwad. 2014. A Proposed System for Hiding Information In Portable Executable Files Based on Analyzing Import Section. IOSR J. Eng., Vol. 04, No. 01, pp. 21–30.

[44] J. Xu, L. Feng, Y. Ye, and Y. Wu. 2012. An Information Hiding Algorithm Based on Bitmap Resource of Portable Executable File. J. Electron. Sci. Technol., Vol. 10, No. 2, pp. 181–184.

[45] A. A. Zaidan, B. B. Zaidan, O. H. Alanazi, A. Gani, O. Zakaria, and G. M. Alam. 2010. Novel approach for high (secure and rate) data hidden within triplex space for executable file. Sci. Res. Essays, Vol. 5, no. 15, pp. 1965–1977.

[46] G. Chen, M. Zhang, and P. Zhang. 2012. The Unlimited Steganographic Capacity Algorithm for Source Code Modification Steganographic Model. J.

Inf. Comput. Sci., Vol. 18, No. 20090322004, pp. 5543–5550.

[47] C. Zhang, H. Peng, X. Long, Z. Pan, and Y. Wu. 2009. A Fragile Software Watermarking for Tamper-Proof. in 2009 Fifth International Conference on Information Assurance and Security, pp. 309–312.

[48] Z. Tian, X. Sun, and H. Yang. 2011. A Scheme of PE Virus Detection Using Fragile Software Watermarking Technique. Int. J. Digit. Content Technol. its Appl., Vol. 5, No. 2, pp. 158–164.

[49] R. M. Karp and M. O. Rabin. 1987. Efficient randomized pattern-matching algorithms. IBM J. Res. Dev., Vol. 31, No. 2, pp. 249 – 260.

[50] S. Schleimer, D. S. Wilkerson, and A. Aiken. 2003. Winnowing: Local Algorithms for Document Fingerprinting. In Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD ’03, 2003, pp. 76–85.

[51] K. Fukushima and K. Sakurai. 2003. A Software Fingerprinting Scheme for Java Using Classfiles Obfuscation. In: 4th International Workshop, WISA 2003 Jeju Island, Korea, August 25-27, 2003 Revised Papers, 2004, pp. 303–316.

[52] A. G. Gounares. 2012. Fingerprinting Executable Code. US 20120317421 A1.

[53] G. Myles and C. Collberg. 2004. Detecting Software Theft via Whole Program Path Birthmarks. In: Proc. Information Security 7th International Conference, ISC 2004, 2004, Vol. 3225, pp. 404–415.

[54] H. Tamada and M. Nakamura. 2004. Design and evaluation of birthmarks for detecting theft of java programs. In: Proceedings of the IASTED International Conference on Software Engineering, 2004, pp. 569–574.

[55] X. Wang, Y. C. Jhi, S. Zhu, and P. Liu. 2009. Detecting software theft via system call based birthmarks. In: Proceedings - Annual Computer Security Applications Conference, ACSAC, 2009, pp. 149–158.

[56] G. Myles and C. Collberg. 2005. K-gram Based Software Birthmarks. In: Proceedings of the 2005 ACM symposium on Applied computing - SAC ’05, 2005, p. 314.

[57] H. Tamada, K. Okamoto, and M. Nakamura. 2004. Dynamic software birthmarks to detect the theft of windows applications. In: International Symposium on Future Software Technology.

[58] H. Tamada and K. Okamoto. 2007. Design and evaluation of dynamic software birthmarks based on api calls. Nara Inst. Sci. Technol. Tech. Rep.

[59] X. Wang, Y.-C. Jhi, S. Zhu, and P. Liu. 2009. Behavior based software theft detection. Proc. 16th ACM Conf. Comput. Commun. Secur. - CCS ’09, p. 280.

[60] H. Tamada, M. Nakamura, A. Monden, and K. Matsumoto. 2003. Detecting the Theft of Programs Using Birthmarks. In: Information Science Technical Report NAIST-IS-TR2003014 ISSN 0919- 9527, 2003, p. 13.