Computer Science and Information Systems 11(3):1055–1069 DOI: 10.2298/CSIS130918064H

A New Detection Scheme of Software Copyright Infringement using Software Birthmark on Windows Systems

Yongman Han[1], Jongcheon Choi[1], Seong-je Cho[1], Haeyoung Yoo[2], Jinwoon Woo[2], Yunmook Nah[3], and Minkyu Park[4]

[1] Dept. of Computer Science, Dankook University
Yongin, Korea, 448-701
{grid_ym, godofslp, sjcho}@dankook.ac.kr

[2] Dept. of Software Science, Dankook University
Yongin, Korea, 448-701
{yoohy, jwwoo}@dankook.ac.kr

[3] Dept. of Applied Computer Engineering, Dankook University
Yongin, Korea, 448-701
ymnah@dankook.ac.kr

[4] Dept. of Computer Engineering, Konkuk University
Chungju, Korea, 380-701
minkyup@kku.ac.kr

Abstract. As software is getting more valuable, unauthorized users or malicious programmers illegally copies and distributes copyrighted software over online service provider (OSP) and P2P networks. To detect, block, and remove pirated software (illegal programs) on OSP and P2P networks, this paper proposes a new filtering approach using software birthmark, which is unique characteristics of program and can be used to identify each program. Software birthmark typically includes constant values, library information, sequence of function calls, and call graphs, etc. We target Microsoft Windows applications and utilize the numbers and names of DLLs and APIs stored in a Windows executable file. Using that information and each cryptographic hash value of the API sequence of programs, we construct software birthmark database. Whenever a program is uploaded or downloaded on OSP and P2P networks, we can identify the program by comparing software birthmark of the program with birthmarks in the database. It is possible to grasp to some extent whether software is an illegally copied one. The experiments show that the proposed software birthmark can effectively identify Windows applications. That is, our proposed technique can be employed to efficiently detect and block pirated programs on OSP and P2P networks.

Keywords: Software birthmark, Import Address Table (IAT), Software piracy, Software identification, Dynamic-Link Library (DLL), Application Programming Interface (API), Windows PE

1. Introduction

Though recent anti-piracy measures monitor Internet for detecting illegal upload or download of music and movies, copyrighted software has been still illegally distributed over Online Service Provider (OSP) and P2P networks. Software piracy is a growing concern in today's competitive world of software. Indeed, many incidents have been reported, and many software developers and copyright holders have been victimized by software theft. The Business Software Alliance (BSA) publishes the yearly study about copyright infringement of software. The Ninth Annual BSA 2011 Piracy Study reported that 57 percent of the world's personal computer users admit to pirating software [2]. The commercial value of all these pirated software rose from $58.8 billion in 2010 to $63.4 billion in 2011. Undoubtedly, software piracy causes severe damages to software industries, stifling not only IT innovation but also job creation across all sectors of the economy. In addition, a recent report of the BSA, "Competitive Advantage: The Economic Impact of Properly Licensed Software", reported that if you use genuine software globally 1% more, there are economic benefits of about $ 73 billion, whereas if you use infringe copyright 1% more, there are economic benefits of about $ 20 billion [3].

To protect the intellectual property for software developers [7], many software protection techniques have been proposed. Among them, software birthmark is a prominent technique. A software birthmark is a unique characteristic, or set of characteristics, that a program inherently has and can be used to identify that program. Existing birthmark schemes have some limitations, though. For example, a static source code-based birthmark [17] requires source code, and is not applicable to binary executable programs. This source code-based birthmark and other birthmarks, such as static executable code-based birthmark [13], dynamic whole program path (WPP)-based birthmark [12], and dynamic API-based birthmark [18], are not resilient to semantics-preserving obfuscation attacks, such as outlining and ordering transformation [8]. Also none of the existing static birthmarks has been evaluated on large-scale programs.

We propose a new software birthmark based on the number and names of *Dynamic Link Libraries* (DLLs) and *Application Programming Interfaces* (APIs) used in Windows applications.

This birthmark can be used to detect the obfuscated Microsoft Windows applications, including large-scale programs, and consequently to detect illegal distribution of copyrighted software over OSP and P2P networks. Windows executable programs have *Portable Executable* (PE) format, and their DLL and API information is stored in a section of PE, *Import Address Table* (IAT). For each application program, the number and names of DLLs and APIs, API call sequence, and a hash value for API call sequence can be inherent to each program and can be used as a unique birthmark. According to the characteristics of the number and names of DLLs and APIs, application programs can be grouped into several categories: Ftp client, Text editor, Media player, Image viewer, Compression tool, Messenger, Cd tool, p2p, etc. A categorization system speeds up search or identification process.

In this paper, we have first construct a birthmark database (DB) which contains the number and names of DLLs and APIs, category information, each hash value of API sequence of a program, and the information indicating that a corresponding program is commercial software or not. Whenever a program, p_i is uploaded or downloaded on

OSP or P2P networks, the identification process of the program consists of four steps: (1) Classifying the p_i into a category, (2) Inspecting the names of DLLs and number of APIs of the p_i targeting only programs classified in the same category, (3) Computing a hash value using the sequence of API calls of the p_i and comparing it with hash values of programs within the identified category, and (4) In case that the categorization in step (1) is failed and then the identification in step (3) is failed too, comparing the hash value of the p_i with the hash values of all programs in the entire DB. If the identified program is commercial, upload or download is not permitted.

The rest of the paper is organized as follows. Section 2 outlines the background and related work. Section 3 describes the proposed software birthmark. In Section 4, we present typical scenario and detailed steps for identifying and filtering copyright infringement software. Section 5 presents the experiment results, and finally we summarize our conclusions and describe future work.

2. Background and Related Work

In this section, we give an overview on Import Address Table (IAT) of the Portable Executable (PE) on Microsoft Windows. The PE is the format of an executable binary on Windows OS. We also explain MD5 hash algorithm and various software birthmark schemes.

2.1. Import Address Table

Microsoft Windows operating systems use the PE format for executable files, object code, and DLLs [11]. The PE format contains dynamic library references for linking, API export and import tables, resource management data and thread-local storage (TLS) data. A PE file consists of a few headers and sections that tell the dynamic linker how to map the file into memory.

When a program is loaded, the Windows loader loads all the DLLs the application uses and maps them into the process address space. A DLL is simply a file that contains one or more pre-compiled functions. That is, each DLL contains pre-compiled implementation code for API functions. The executable file lists all the functions it requires from each DLL. This loading and joining is accomplished by using the IAT. The IAT is a table of function pointers filled in by the Windows loader as the DLLs are loaded.

The IAT is a lookup table when the application is calling a function from a different module. It can be in the form of both import by ordinal and import by name [11]. The IAT of a PE file is used to store virtual addresses of functions that are imported from external PE files. From the IAT, we can obtain the feature information of the program, such as the number of DLLs, the names of DLLs, and the names of API functions in each DLL.

2.2. MD5 (Message-Digest algorithm5) Hash Function

MD5 hash function receives a message of arbitrary length as input and output a 128 bit value. This function is widely used to check the integrity of an original executable file. It also can be used to identify specific software. However, a hashing function generates a completely different value from one bit change (Fig. 1).

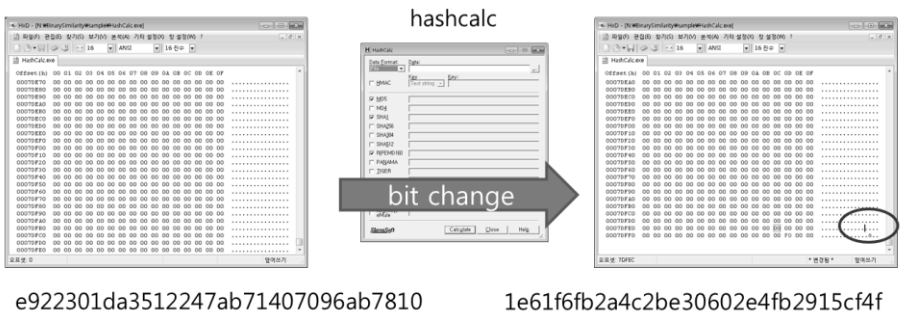

hashcalc
bit change
e922301da3512247ab71407096ab7810
1e61f6fb2a4c2be30602e4fb2915cf4f

Fig. 1. A one bit change can generate an entirely different hash value

2.3. Related Work

A source code-based birthmark uses names of variables and functions [4]. This birthmark, however, no longer exists after compilation without special handling. Given only an executable file, we cannot use this birthmark for its original purpose.

Because of this limitation, many researchers are studying on API-based or system call-based birthmarks. These birthmarks are intact through compilation and can be used for detecting software theft and computer forensics.

Existing birthmarks can be classified into two categories. Static birthmarks extract statically available information in the program source code or executable files [4,9,13,17,20], for example, the types or initial values of the fields. Dynamic birthmarks, in contrast, rely on information gathered from the execution of a program [1,10,12,18].

Tamada et al. [17] proposed four types of static birthmark: constant values in field variables, sequence of method calls, inheritance structure, and used classes. All the four types are vulnerable to obfuscation techniques, such as code removal or splitting of variables [12]. In addition, their technique needs to access the source code and only works for an object-oriented programming language, such as Java.

Myles and Collberg proposed K-Gram-based birthmark, a static technique, which uniquely identifies a program through instruction sequences [13]. Instruction (opcode) sequences of length k are extracted from a program, and k-gram techniques, which were used to detect the similarity of documents [15], are used for the opcode sequence. The k-gram static birthmark is still fragile to some obfuscation methods, such as statement reordering, invalid instruction insertion, and compiler optimization.

Myles and Collberg presented another dynamic birthmark called a whole program path (WPP) and evaluated its performance on a Java program [12,14]. WPP is originally used to represent the dynamic control flow graphs (DCFGs) of a program. It collects all the compact DCFGs and regards them as a program's birthmarks. However, a WPP may not work for large-scale programs because of the overwhelming volume of WPP traces. Also, it is vulnerable to program optimization, such as loop transformations and inline functions.

Tamada et al. [18] introduced two types of dynamic birthmark for Windows applications: sequence of API function calls and frequency of API function calls. The sequence and frequency of Windows API calls are recorded during the execution of a program. Shuler and Dallmeier [16] presented a dynamic birthmark based on the extraction of API call sequence sets during program execution. API birthmarks are more robust to obfuscation than WPP birthmarks [19]. However, dynamic birthmarks need program executions which are dependent on user interactions, inputs, and environments.

Wang et al. [19] proposed two types of system call birthmark: system call short sequence birthmark and input-dependent system call subsequence birthmark. System call-based birthmarks can be platform-independent and are more robust to counter-attacks than API-based ones. They also need a program execution. Moreover, there are no easy ways to record system call traces of each application during program execution on Microsoft Windows systems.

Choi et al. [6] suggested a static API birthmark for Windows. Their birthmark is a set of possible API calls which are statically extracted by analyzing disassembled code. They did not use DLL information, which can be easily obtained from the IAT.

In our previous work [5], we have proposed the similar software birthmark to one proposed in this paper in order to identify each program. However, the previous software birthmark did not consider the sequence of API calls and its hash value, thus had some limitation to efficiently identify some programs of different versions. In addition, our previous work did not use software classification, and then had to compare the birthmark of a given program with all birthmarks in a birthmark database through the four steps. In this paper, we introduce (1) classification scheme to group some similar programs into a same category, and (2) API call sequence of a program and its hash value.

Current birthmarks are limited in their capabilities: some solutions are not strong enough to adequately prevent software theft, some cause significant performance degradation for large-scale programs, and some need program execution or work only for Java programs.

3. The Proposed Software Birthmark

The proposed software birthmark includes the following features (Fig. 2):

– number of DLLs and their names
– number of APIs and their names
– sequence of API calls

We extract the first two pieces of information from the IAT of the executable file.

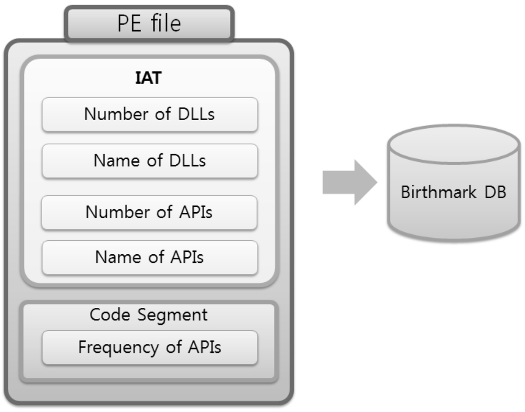
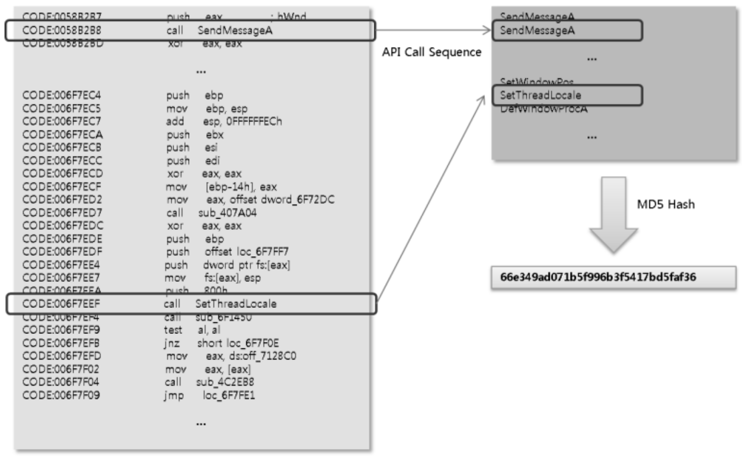


Fig. 2. The proposed software birthmark of Windows PE format file

Sequence of API calls can be obtained from the code segment of the executable file. The executable file is disassembled and sequence is extracted from it. We, then, calculate MD5 hash value on it (Fig. 3).



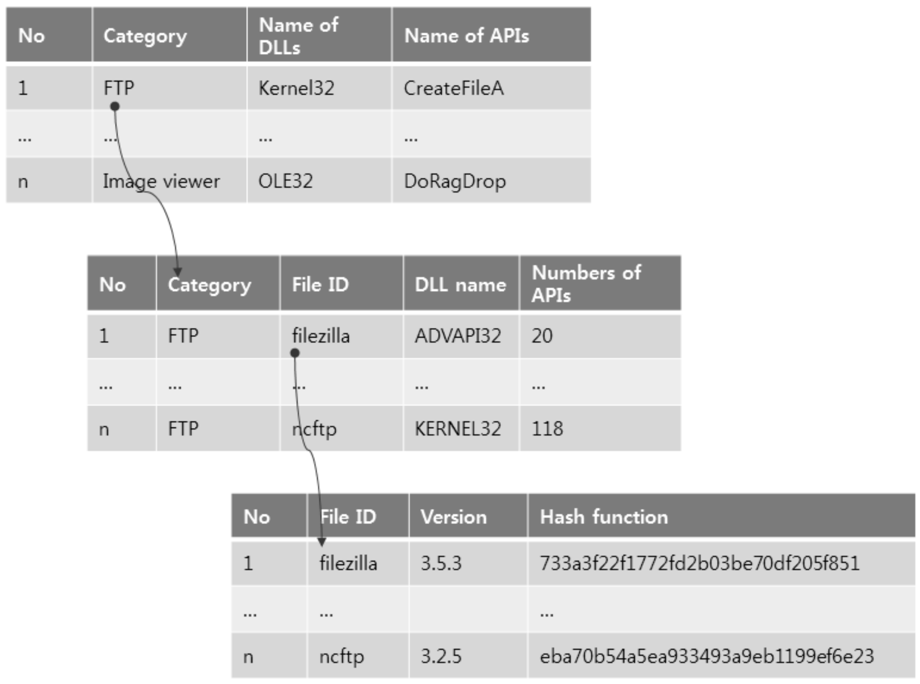
```
CODE:0058B2B7          push   eax            ; hWnd
CODE:0058B2B8          call   SendMessageA
CODE:0058B2BD          xor    eax, eax

                       ...

CODE:006F7EC4          push   ebp
CODE:006F7EC5          mov    ebp, esp
CODE:006F7EC7          add    esp, 0FFFFFFECh
CODE:006F7ECA          push   ebx
CODE:006F7ECB          push   esi
CODE:006F7ECC          push   edi
CODE:006F7ECD          xor    eax, eax
CODE:006F7ECF          mov    [ebp-14h], eax
CODE:006F7ED2          mov    eax, offset dword_6F72DC
CODE:006F7ED7          call   sub_407A04
CODE:006F7EDC          xor    eax, eax
CODE:006F7EDE          push   ebp
CODE:006F7EDF          push   offset loc_6F7FF7
CODE:006F7EE4          push   dword ptr fs:[eax]
CODE:006F7EE7          mov    fs:[eax], esp
CODE:006F7EEA          push   800h
CODE:006F7EEF          call   SetThreadLocale
CODE:006F7EF4          call   sub_6F1450
CODE:006F7EF9          test   al, al
CODE:006F7EFB          jnz    short loc_6F7F0E
CODE:006F7EFD          mov    eax, ds:off_7128C0
CODE:006F7F02          mov    eax, [eax]
CODE:006F7F04          call   sub_4C2EB8
CODE:006F7F09          jmp    loc_6F7FE1

                       ...
```

```
SendMessageA
SendMessageA
...
SetWindowPos
SetThreadLocale
DefWindowProcA
...
```

```
66e349ad071b5f996b3f5417bd5faf36
```

Fig. 3. How to caculate MD5 hash value from sequence of API calls

We store all this information to birthmark DB. The Schema of Feature Database is shown in Fig. 4. This database is a relational database and has several tables for DLL names, API names, and hash values. These tables are File information table, DLL information table, and API information table. The tables can be accessed using a file name and a DLL name.



No	Category	Name of DLLs	Name of APIs
1	FTP	Kernel32	CreateFileA
...	...	...	...
n	Image viewer	OLE32	DoRagDrop

No	Category	File ID	DLL name	Numbers of APIs
1	FTP	filezilla	ADVAPI32	20
...	...	...	...	...
n	FTP	ncftp	KERNEL32	118

No	File ID	Version	Hash function
1	filezilla	3.5.3	733a3f22f1772fd2b03be70df205f851
...	...		...
n	ncftp	3.2.5	eba70b54a5ea933493a9eb1199ef6e23

Fig. 4. The schema of the birthmark DB

You can see more details about this approach in our preliminary version of this paper [5].

4. Software Filtering using the Software Birthmark

4.1. Identifying and Filtering Overview

When a user tried to upload an application to an OSP, the OSP stores it at the temporary folder and asks the checking module that implements our proposed detection scheme whether it is commercial software distributed illegally. The checking module first extracts the software birthmark from the executable files of the application. The module, then, compares DLL and API information of the birthmark with category information in the birthmark DB to categorize it. After categorization, the module compare with all applications in the identified category using number of DLLs, their names, number of APIs and their names. If the module cannot identify the application, the module compares the hash value with the hash values of all applications in the same category. If the applications are not identified, the module compares the hash value with all hash values in the birthmark DB.

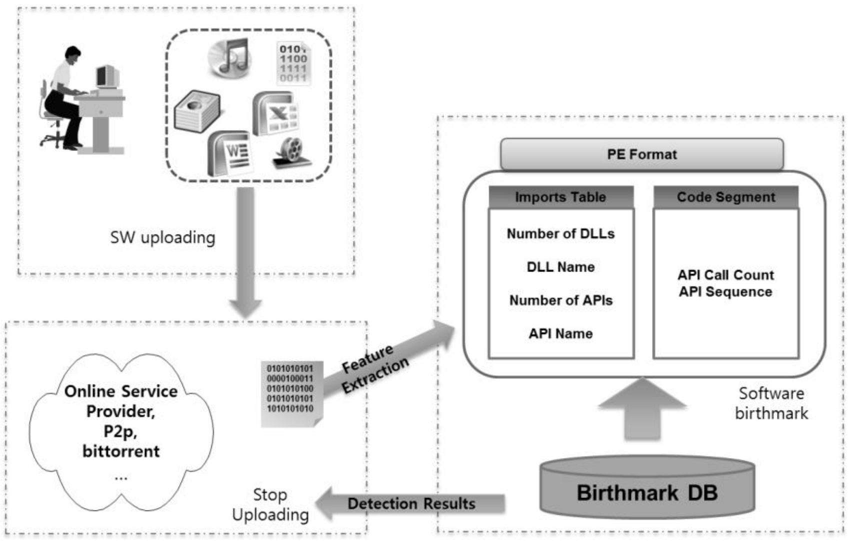


Fig. 5. The software identification and filtering process

If the application is illegally distributed commercial one, then the OSP stops the uploading procedure and delete the application. If the module cannot identify the application, the OSP inserts its software birthmark into the birthmark DB.

4.2. Detailed Steps

We describe the detailed identifying procedure. We denote the application being uploaded as p_i.

Step 1: Classifying the pi into a category. Using extracted software birthmark, we tries to identify a general kind of application. For example, if an application has software birthmark that appears in text editor, we can conclude the application may be some kind of text editor. We select 8 categories to identify, such as FTP client, Media player, Image viewer, etc. We think those categories include most representative application distributed via the Internet. This categorization helps to reduce the number of applications in the birthmark DB to compare. Software categorization, thus, can decrease comparison time when the size of the database is very large. If software cannot be identified, go to Step 3.

Step 2: Inspecting the names of DLLs and number of APIs of the pi targeting only programs classified in the same category. After the previous categorization, we compare names of DLL and the number of API functions used in the whole executable to the application in the same category, respectively. If the programs are not identified, go to Step 3.

Step 3: Computing a hash value using the sequence of API calls of the pi and comparing it with hash values of programs within the identified category. We extract the sequence of API calls from the code segment of the executable and input to the MD5 hash function. MD5 generate the 128 bit hash value. This hash value is compared to the hash values of the application belonging to the previously identified software category. We think this sequence may not change even against semantic preserving transformation attack. If the programs are not identified, then go to Step 4.

Step 4: comparing the hash value of the pi with the hash values of all programs in the entire DB. If an application is not identified yet, there may be some problems with categorization in Step 1. Therefore, we compare the hash value to the hash values stored in entire birthmark DB.

5. Experiments and Evaluation

5.1. Target Applications

To evaluate the effectiveness of our birthmark, we conduct an experiment using sample programs listed in Table 1. Sample programs are chosen in various categories like FTP clients, text editors, media players, etc.

Table 1. Sample applications

Group	No.	Program	Version	Size (Kb)
FTP Client	(1)	Alftp	5.3.2	4,109
	(2)	Ncftp	3.2.5	300
	(3)-a	Filezilla	3.5.3	7,994
	(3)-b		3.5.2	7,993
	(3)-c		3.4.0	7,463
	(4)-a	WinSCP	4.3.9	6,329
	(4)-b		4.3.8	6,325
	(4)-c		4.0.4	4,878
Text Editor	(5)	Editplus	3.20	1,787
	(6)	Eclipse	1.4.9	52
	(7)	EXPAD	0.4	845
	(8)-a	AkelPad	4.7.7	357
	(8)-b		4.7.6	357
	(8)-c		4.5.6	321
	(9)-a	Notepad++	6.1.5	1,584
	(9)-b		6.1.4	1,584
	(9)-c		5.8.0	1,308
Media Player	(10)	Alshow	2.02	117
	(11)	Coolplayer	2.19	3,817

	(12)	GOM Player	2.1.43	3,948
	(13)	KMPlayer	3.3.0	7,521
	(14)	Loongplayer	1.01	920
	(15)	Mplayerc	6.4.9.1	4,308
	(16)	Potplayer	1.51	180
	(17)	Winamp	5.6.3	2,156
	(18)-a	Mixxx	1.10.1	3,058
	(18)-b		1.10.0	3,028
	(18)-c		1.07.2	2,132
Image viewer	(19)	Alsee	6.8	6,960
	(20)	Imagine	1.0.8	17
	(21)	Xnview	1.99	4,624
Compress Tools	(22)	Alzip	8.53	2,855
	(23)	Backzip	5.03	1,920
	(24)	Peazip	4.6.1	4,023
	(25)	TUGZip	3.5	3,361
	(26)-a	7zFM	9.22	411
	(26)-b		9.20	412
	(26)-c		9.04	383
messenger	(27)	Pidgin	2.10.6	49
	(28)	Psi	0.15	6,869
	(29)	RetroShare	0.54	14,340
Cd tool	(30)	CDspace7 lite	1.02	2,191
	(31)	Dtlite	4.41	4,796
p2p	(32)	Emul	5.0	5,624
	(33)	Youdonkey	2.35	240

5.2. Identifying the Target Applications

The overall comparison and identification results are shown in Table 2.

Table 2. Application identification results

Group	No.	Step 1	Step 2	Step 3	Step 4
FTP Client	(1)	FTP/Media	Identified		
	(2)	FTP	Identified		
	(3)-a	FTP/Text	3/17	Identified	
	(3)-b	FTP/Text	3/17	Identified	
	(3)-c	FTP/Text	3/17	Identified	
	(4)-a	FTP/Media	3/19	Identified	
	(4)-b	FTP/Media	3/19	Identified	
	(4)-c	FTP/Media	3/19	Identified	
Text Editor	(5)	Text/Media	Identified		
	(6)	Text/Zip/p2p	Identified		
	(7)	Text/Zip/Msg	Identified		
	(8)-a	Text	2/9	Identified	

	(8)-b	Text	2/9	Identified	
	(8)-c	Text	Identified		
	(9)-a	Text	3/9	Identified	
	(9)-b	Text	3/9	Identified	
	(9)-c	Text	3/9	Identified	
Media Player	(10)	Media	Identified		
	(11)	Media	Identified		
	(12)	Text/Media	Identified		
	(13)	Media	Identified		
	(14)	Media/Zip/Msg	Identified		
	(15)	Media	Identified		
	(16)	Media	Identified		
	(17)	Media/Zip	Identified		
	(18)-a	Media/Zip	2/11	Identified	
	(18)-b	Media/Zip	2/11	Identified	
	(18)-c	Media/Zip	Identified		
Image viewer	(19)	Text/Media/ Image	Identified		
	(20)	Media/Image	Identified		
	(21)	Image	Identified		
Compress Tools (zip)	(22)	Text/Media/Zip	Identified		
	(23)	Zip	Identified		
	(24)	Text/Media/ Image	0/23	0/23	Identified
	(25)	FTP/Text	0/17	0/17	Identified
	(26)-a	Zip	2/2	Identified	
	(26)-b	Zip	2/2	Identified	
	(26)-c	Zip	Identified		
messenger	(27)	Msg	Identified		
	(28)	Msg	Identified		
	(29)	Zip/Msg	Identified		
Cd tool	(30)	Cd tool	Identified		
	(31)	Cd tool	Identified		
p2p	(32)	p2p	Identified		
	(33)	p2p	Identified		

The Step 1 column of Table 1 represents the identified category after step 1 completes. Categorization is based on the assumption that programs in the same category use common DLLs and APIs. If an application is not clearly determined and seems to belong to two or more categories simultaneously, we compare it to all applications in both categories.

In Step 2, we try to identify only one application and uses DLL names and the number of APIs used. In Step 3, we extract the sequence of API calls from the disassembled code and generate MD5 hash value on it (Fig. 5). This hash value is compared to hash values of applications in the same category identified in Step 1.

If an application is not identified after Setp3, there are two cases we can think of.

Case 1: A new application. In this case, there is no information of the application considered in birthmark DB.

Case 2: Categorization failure. Step 1 fails to categorize an application. In our experiment, Peazip and Tugzip are such a case. In this case we compare the hash value of an application to all hash values of applications in birthmark DB.

After Step 2, we can identify most applications, but cannot identify applications with small difference. After Step 3, those applications can be identified and so MD5 hash function is effective for applications with small difference.

5.3. Measuring the Time to Identify an Application

We experimented with applications described in section 5.2 and obtained a detection accuracy of 95.56%. Since the difference between measured times was about 50ms, we repeated 3 times for one program and calculated the average time for each program. We calculate the average time for all programs by summing up all the average times calculated above and dividing the sum by the number of all programs. Fig.6 shows minimum, average, and maximum detection time. The minimum and maximum time equals to the smallest and largest average time, respectively. Longplayer is identified in the shortest time, 1981ms, because the number of API functions and DLL files used was small. The Peazip, on the other hand, was detected after the longest time has passed, 3345ms. In the case of Peazip, we need to complete step4 to identify. The average time is 2545ms, and most programs were discernible after Step 2.

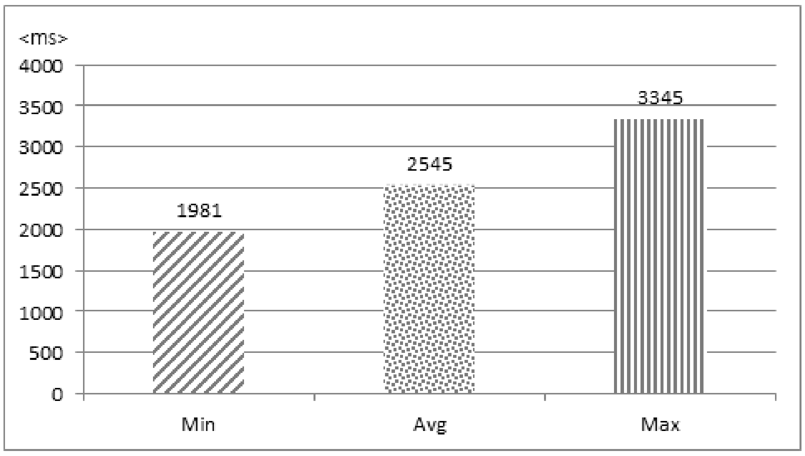


Fig. 6. The time required for identifying an application

6. Conclusion and Future Work

To detect software theft or piracy, a birthmark relies on the inherent characteristics of an application, which can be used show that one program is a copy of another. In this paper, we have proposed a new static birthmark scheme using the notion of Import Address Table, which can be used to identify Windows executable files, and MD5 hash values from sequence of API calls. Our birthmark is obtained by analyzing a Windows PE executable file and disassembling the PE file. We store this birthmark into a birthmark database and use it to compare the features of programs in concern.

We also use MD5 hash function on a sequence of API calls of an application. The sequence is extracted from the disassembled code of the application. This sequence is strong against the semantic preserving transformation attack.

We are working on ways to improve the efficiency of detecting illegal software and to elaborate comparisons with frequently used DLLs.

Acknowledgements. This research project was supported by Ministry of Culture, Sports and Tourism (MCST) and from Korea Copyright Commission in 2013, and by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014-H0301-14-1023) supervised by the NIPA (National IT Industry Promotion Agency).

References

1. Bai, Y., Sun, X., Sun, G., Deng, X., Zhou, X.: Dynamic K-gram based Software Birthmark. In Proceedings of 19th Australian Conference on Software Engineering. IEEE Computer Society, 644-649. (2008)
2. BSA: Shadow Market: 2011 BSA Global Software Piracy Study. Business Software Alliance, (2012)
3. BSA: Competitive Advantage: The Economic Impact of Properly Licensed Software. Business Software Alliance, (2013)
4. Burrows, S., Tahaghoghi, S., Zobel, J.: Efficient plagiarism detection for large code repositories. Software-Practice and Experience, Vol. 37, No. 2, 151-175. (2007)
5. Choi, J., Han, Y., Cho, S., Yoo, H., Woo, J., Park, M.: A Static Birthmark for MS Windows Applications Using Import Address Table. In Proceedings of the 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing 2013. IEEE, 129-134. (2013)
6. Choi, S., Park, H., Lim, H., Han, T.: A Static Birthmark of Binary Executables Based on API Call Structure. In Proceedings of 12th Asian Computing Science Conference. Springer, 2-16. (2007)
7. Collberg, C., Thomborson, C.: Software Watermarking: Models and Dynamic Embeddings. In Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages. ACM, 311-324. (1999)
8. Kim, H., Khoo, W. M., Lio, P.: Polymorphic Attacks against Sequence-based Software Birthmarks. In Proceedings of 2nd Software Security and Protection Workshop. ACM. (2012)
9. Lim, H., Park, H., Choi, S., Han, T.: A Static Java Birthmark Based on Control Flow Edges. In Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference. IEEE Computer Society, 413-420. (2009)

10. Lu, B., Liu, F., Ge, X., Liu, B., Luo, X.: A Software Birthmark Based on Dynamic Opcode n-gram. In Proceedings of the First IEEE International Conference on Semantic Computing. IEEE Computer Society, 37-44. (2007)
11. Microsoft.: Microsoft Portable Executable and Common Object File Format Specification. Revision 8.2. (2010)
12. Myles, G., Collberg, C.: Detecting Software Theft via Whole Program Path Birthmarks. In Proceedings of 7th International Information Security Conference. Springer, 404-415. (2004)
13. Myles, G., Collberg, C.: K-gram based software birthmarks. in Proceedings of the 2005 ACM Symposium on Applied Computing. ACM, 314-318. (2005)
14. Myles, G.: Software Theft Detection Through Program Identification. PhD thesis. Department of Computer Science. The University of Arizona. (2006)
15. Schleimer, S., Wilkerson, D., Aiken, A.: Winnowing: Local Algorithms for Document Fingerprinting. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data. ACM, 76-85. (2003)
16. Schuler, D., Dallmeier, V.: Detecting Software Theft with API Call Sequence Sets. In Proceedings of the 8th Workshop on Software Reengineering. ACM German Chapter, 56-57. (2006)
17. Tamada, H., Nakamura, M., Monden, A., Matsumoto, K.: Design and Evaluation of Birthmarks for Detecting Theft of Java Programs. In Proceedings of IASTED International Conference on Software Engineering. ACTA Press, 569-575. (2004)
18. Tamada, H., Okamoto, K., Nakamura, M., Monden, A., Matsumoto. K.: Dynamic Software Birthmarks to Detect the Theft of Windows Applications. In Proceedings of International Symposium on Future Software Technology. Software Engineers Association, 280-285. (2004)
19. Wang, X., Jhi, Y., Zhu, S., Liu, P.: Detecting Software Theft via System Call Based Birthmarks. In Proceedings of 25th Annual Computer Security Applications Conference. IEEE Computer Society, 149-158. (2009)
20. Xie, X., Liu, F., Lu, B., Chen, L.: A Software Birthmark Based on Weighted K-gram. In Proceedings of IEEE International Conference on Intelligent Computing and Intelligent Systems. IEEE, 400-405. (2010)

Yongman Han is doing a Ph.D. in Computer Science from University of Dankook, Korea in 2012. He has a master's degree from Dankook University. His research interests include software similarity, software theft, software engineering, software quality. He has authored and co-authored several journals and conference papers.

Jongcheon Choi is doing a Ph.D. in Computer Science from University of Dankook, Korea in 2005. He has a master's degree from Dankook University. His research interests include computer security, software theft, system software, software Protection. He has authored and co-authored several journals and conference papers.

Seong-je Cho received the B.E., the M.E. and the Ph.D. in Computer Engineering from Seoul National University in 1989, 1991 and 1996 respectively. He was a visiting scholar at Department of EECS, University of California, Irvine, USA in 2001, and at Department of Electrical and Computer Engineering, University of Cincinnati, USA in 2009 respectively. He is a Professor in Department of Computer Science, Dankook University, Korea from 1997. His current research interests include computer security, mobile security, operating systems, and software protection.

Haeyoung Yoo received the Ph.D. degree in Department of Computer Engineering, Ajou University in 1994. He is now a Professor in Department of Software Science, Dankook University, Korea. And he is now a vice-chairman of Korea Copyright Commission. His research interests include software development methodology, content technology policy and development, software testing and web engineering. He has authored and co-authored several journals and conference papers and software engineering textbook.

Jinwoon Woo received the Ph.D. degree in Department of Computer science, University of Minnesota, USA in 1990. He is now a Professor in Department of Software Science, Dankook University, Korea. His research interests include algorithm, information security, software assurance. He has authored and co-authored several journals and conference papers.

Yunmook Nah received the Ph.D. degree in Department of Applied Computer Engineering, Seoul National University in 1993. He is now a Professor in Department of Computer Engineering, Dankook University, Korea. His research interests include database, data modeling, large distributed database. He has authored and co-authored several journals and conference papers and database textbook.

Minkyu Park is the corresponding author of this paper. He received the B.E. and M.E. degree in Computer Engineering from Seoul National University in 1991 and 1993, respectively. He received Ph.D. degree in Computer Engineering from Seoul National University in 2005. He is now an Associate Professor in Konkuk University, Korea. His research interests include operating systems, real-time scheduling, embedded software, computer system security, and HCI.

Received: September 18, 2013; Accepted: January 21, 2014.