An Efficient Categorization of the Instructions Based on Binary Excutables for Dynamic Software Birthmark

Donghoon Lee, Younsung Choi, Jaewook Jung, Jiye Kim, and Dongho Won

***Abstract*—Software birthmark is a unique characteristic of program extracted from a program without source code. Through the comparison of original program and modified program, code similarity can be measured. Furthermore, birthmark can be used to measure the similarity of existing program to detect code theft or malware. Software birthmark can be mainly divided into static method and dynamic method. In the related works using dynamic method, birthmark was extracted by using API function name, call frequency, grammar structure, opcode, etc. If birthmark is extracted through API function name or call frequency, resilience can be increased but it could cause false-positive in similarity. In addition, extraction method using grammar structure or opcode could increase similarity but it decreases resilience, thereby causing different extraction result even for program with same structure. This paper proposes a method that can simultaneously satisfy resilience and uniqueness by reflecting unique characteristics while maintaining the meaning of instruction through the categorization according to instruction function and the removal of consecutive duplication for dynamic software birthmark, which will also be verified through experiment.**

***Index Terms*—Dynamic software birthmark, code theft detection, information security, dynamic analysis.**

I. Introduction

Definition of `birthmark' is a mark or speckle on the body (from birth). In the area of computer science, this terms has been used to indicate a unique characteristic of program. Accordingly, software birthmark indicates unique information of program extracted from program execution file. The purpose of extracting birthmark from program is to measure the similarity between programs. Accuracy of similarity changes based on how well unique characteristic of program has been reflected. However, resilience decreases when only uniqueness is concentrated. Resilience of birthmark refers to the fact that similarity should be same when program written with same source code has been built in different compile. Accordingly, effective birthmark extraction method simultaneously satisfies uniqueness and resilience. Software birthmark was first proposed to detect software code theft. Currently, it is also being used in digital forensic areas for finding similarity in metamorphic malware by extracting birthmark from various viruses, Trojan horse, worms programs, etc [1]-[3].

Software birthmark can be divided into static birthmark and dynamic birthmark according to extraction method. Static birthmark is extracted in file state without executing program code. Accordingly, it is dependent upon execution file format such as x86 PE(portable executable). As for the advantage of static birthmark, overhead is less during extraction as it targets program in file state and it is not dependent upon analysis tool. Accordingly, it allows real-time automation system design. However, it is difficult to extract birthmark through automated method since recovery of code becomes difficult when packing or code obfuscation has been applied in program [4]-[10]. That's why related works on static birthmark have been conducted under the supposition that packing or obfuscation has not been applied. In addition, compilers that create byte code such as JAVA can extract relatively more accurate static birthmark compared to binary execution file, but it is difficult to extract by accurately analyzing code since accurate distinction between code and data is not possible with binary execution file. That's why dynamic birthmark method has been studied to extract birthmark in executed state of program [5], [6], [8], [9].

Dynamic software birthmark refers to a method of extracting unique information from program in executed state. Since information is extracted in executed state of program in dynamic method, relatively more accurate unique information can be extracted compared to static method. It also has the advantage of being able to analyze even obfuscated execution file. However, it is difficult to design through real-time automation system due to its more overhead that occurs duri- ng analysis process compared to static extraction method [11].

As for the elements for extracting dynamic birthmark, API function name, opcode, etc. are used to measure call frequency for extraction. In addition, there is a method of extracting through other elements such as grammar structure or branch processing sentence structure. Although call frequency such as API function name or opcode satisfies resilience, it does not effectively reflect the unique characteristic of program. In addition, there is a need to increase the specificity of birthmark since false-positive could occur in similarity measurement since functions performed accordingly have been set with API functions provided by OS.

The approach through function name or syntax frequency cannot be distinguished from static method even though it is dynamic method. However, using instruction sequence analysis for extraction is not effective since the extraction scope becomes massive. Accordingly, this paper proposes extraction through instruction categorization to simultaneously satisfy the resilience and uniqueness of birthmark.

The composition of this paper is as follows. Section II

Manuscript received April 5, 2014; revised June 24, 2014.

The authors are with the School of Information and Communication Engineering, Sungkyunkwan University, Korea (e-mail: dhlee@security.re.kr, yschoi@security.re.kr, jwjung@security.re.kr, jykim@security.re.kr, dhwon@security.re.kr).

discusses software birthmark and other works related to this paper. Section III introduces tools for extracting birthmark in executed state program, and proposes birthmark extraction method through instruction categorization. Section IV verifies the approach through experiment, and we conclude in Section V.

II. Related Works

Software birthmark is a unique characteristic for identifying program without source code with binary alone. The representative methods of detecting the similarity of code using execution file are software birthmark and software watermarking. With software watermark, similarity is determined by installing information in execution file in the form of watermark for detecting code theft such as copyright and ownership. With software birthmark, code is analyzed for extraction without including additional information in execution file, unlike software watermarking [12]. With dynamic birthmark, it is difficult to extract automated birthmark since contents of birthmark is extracted while executing program. In addition, contents of birthmark change according to the execution environment or input value. The advantage of dynamic birthmark is that it is strong against code obfuscation with superior accuracy of analysis result compared to static birthmark. The representative dynamic birthmark include Tamada's “Dynamic software birthmarks to detect the theft of windows applications", Myles's “Detecting Software Theft via Whole Program Path Birthmarks" and Schuler's “Dynamic Java API Birthmark" [1], [13], [14]. In related works, call frequency of particular syntax was extracted in the program default state. Such method was proposed strictly as a method of bypassing code obfuscation without differentiation from static extraction method [15]. The advantage of dynamic analysis is in the executed instruction itself. With Lianhong's “Instruction-words based Software Birthmark”, birthmark is extracted through the frequency of instruction word [16]. In addition, Bin's “A Software Birthmark Based on Dynamic Opcode n-gram” is also a method of extracting birthmark using opcode of program [2]. The birthmark extraction method using instruction decreased resilience by only reflecting unique characteristics of program while causing overhead by analyzing massive amounts of instruction sequence.

For the purpose of improving the issues of related works, a method of reducing the volume of birthmark extraction contents while not damaging the meaning of sequence is proposed in this study by removing consecutive duplication through the categorization of instruction.

III. Proposed Method

In this section, a method of extracting birthmark in executed state program is proposed.

A. DBI Tools

In this paper, DBI (dynamic binary instrumentation) tool is used to extract birthmark of program in executed state. The DBI refers to a technology for inserting random code in executed program to collect execution information with special purposes (debugging, monitoring, statistic, etc.).

DBI tools include Dynamic RIO, Pin and Valgrind for LINUX OS. As shown in Table I, Dynamic RIO and Pin support Windows, Linux and Max OS. Valgrind only supports Linux environment [17]. Pin tool, in particular, is an official DBI tool of Intel Corporation. In this paper, Pin will be used as a DBI tool to extract birthmark of binary program developed through C, C++ in Windows OS environment [18], [19].

Pin provides efficient instrumentation by using a just-in-time (JIT) compiler to insert and optimize code. It supports the IA32, EM64T, Itanium, and ARM architectures running Linux.

TABLE I: DBI Tools for Operation Systems

DBI	WIN	LINUX	MAX OS X
PIN	○	○	○
Dynamic RIO	○	○	○
Valgrind		○	

Pin compiles from one ISA directly into the same ISA (e.g., IA32 to IA32, ARM to ARM) without going through an intermediate format, and the compiled code is stored in a software-based code cache.

The Pin API makes it possible to observe all the architectural state of a process, such as the contents of registers, memory, and control flow. It uses a model similar to ATOM, where the user adds procedures (as known as analysis routines in ATOM’s notion) to the application process, and writes instrumentation routines to determine where to place calls to analysis routines. The arguments to analysis routines can be architectural state or constants. Pin also provides a limited ability to alter the program behavior by allowing an analysis routine to overwrite application registers and application memory [11].

- **Routine** (RTN_AddInstrumentFunction)
  Add a function used to instrument at routine granularity.
- **Image** (IMG_AddInstrumentFunction)
  Use this to register a call back to catch the loading of an Image.
- **Trace** (TRACE_AddInstrumentFunction)
  Add a function used to instrument at trace granularity.
- **Instrumentation** (INS_AddInstrumentFunction)
  Add a function used to instrument at instruction granularity.

In this paper, experiment will be conducted with module developed with API of trace and routine of Pin. With routine function, the state of program placed in virtual memory can be extracted. With trace function, extraction can be conducted from the entry point of program in instruction unit. In Chapter 4, we experimented the difference of volume from the birthmark extracted through two methods.

B. Categories of Functional Instructions

In the "Intel 64 and IA-32 Architectures Software Developer’s Manual" provided by Intel Corporation, it explains in details about instruction. Its categorization is as follows according to the characteristics of instructions.

TABLE II: CATEGORIES OF INSTRUCTIONS TO THE INTEL'S MANUAL AND PIN

GENERAL CATEGORY	PIN's CATEGORY	INSTRUCTIONS
Data Transfer	DATAXFER	mov, movsx, movzx, movd, movdqa, *etc.*
	POP	pop, popad, popfd, *etc.*
	PUSH	push, pushfd, pushad, *etc.*
Binary Arithmetic	BINARY	add, cmp, sub, imul, inc, dec, neg, adc, *etc.*
Decimal Arithmetic	DECIMAL	aas, aad, aam, daa, das, *etc.*
Logical	LOGICAL	xor, or, test, and, not, pxor, pandn, andpd, orpd
Shift and Rotate	ROTATE	rcr, ror, rol, rcl, *etc.*
	SHIFT	shl, sar, shr, shrd, shld, *etc.*
Bit and Byte	BITBYTE	btr, bts, setz, setnz, bt, bsf, *etc.*
Control Transfer	COND_BR	jnz, jz, jbe, jns, jnl, jb, jle, jnb, loop, jecxz, *etc.*
	UNCOND_BR	jmp
	CALL	call
	RET	ret, iret
	INTERRUPT	int, int3, int0, int1, bound, *etc.*
String	STRINGOP	lodsd, movsd, rep, stosd, scasb, scasd, lodsb, movsw, *etc.*
I/O	IO	in, out
	IOSTRINGOP	outsb, insb, outsd, insd
Enter and Leave/Miscellaneous	MISC	lea, leave, cupid, pause, enter, xlat, sfence, *etc.*
	NOP	nop
Flag Control (EFLAG)	FLAGGOP	std, cld, lahf, cli, sti, sahf, cmc, clc, *etc.*
Segment Register	SEGOP	les, lds

1) *Data Transfer Instructions*: The data transfer instructions move data between memory and the general-purpose and segment registers. They also perform specific operations such as conditional moves, stack access, and data conversion.
   instructions: *mov, push, pop, etc.*
2) *Binary Arithmetic Instructions*: The binary arithmetic instructions perform basic binary integer computations on byte, word, and doubleword integers located in memory and/or the general purpose registers.
   instructions : *add, sub, mul, div, etc.*
3) *Decimal Arithmetic Instructions*: The decimal arithmetic instructions perform decimal arithmetic on binary coded decimal (BCD) data.
   instructions : *daa, das, aaa, aas, etc.*
4) *Logical Instructions*: The logical instructions perform basic AND, OR, XOR, and NOT logical operations on byte, word, and doubleword values.
   instructions : *and, or, xor, not.*
5) *Shift and Rotate Instructions*: The shift and rotate instructions shift and rotate the bits in word and doubleword operands.
   instructions: *shr, shrd, shld, ror, rol, etc.*
6) *Bit and Byte Instructions*: Bit instructions test and modify individual bits in word and doubleword operands. Byte instructions set the value of a byte operand to indicate the status of flags in the EFLAGS register.
   instructions : *bt, bts, btr, sets, setns, test, etc.*
7) *Control Transfer Instructions*: The control transfer instructions provide jump, conditional jump, loop, and call and return operations to control program flow.
   instructions : *jmp, je, jz, jne, jnz, loop, call, ret, etc.*
8) *String Instructions*: The string instructions operate on strings of bytes, allowing them to be moved to and from memory.
   instructions: *movs, cmp, scas, lods, stos, rep, etc.*
9) *I/O Instructions*: These instructions move data between the processor's I/O ports and a register or memory.
   instructions : *in, out, ins, outs, etc.*
10) *Enter and Leave Instructions*: These instructions provide machine-language support for procedure calls in block-structured languages.
    instructions : *enter, leave, etc.*
11) *Miscellaneous Instructions*: The miscellaneous instructions provide such functions as loading an effective address, executing a "no-operation," and retrieving processor identification information.
    instructions : *lea, nop, cupid, movbe, etc.*
12) *Flag Control (EFLAG) Instructions*: The flag control instructions operate on the flags in the EFLAGS register.
    instructions : *stc, clc, cmc, cld, std, sti, cli, etc.*
13) *Segment Register Instructions*: The segment register instructions allow far pointers (segment addresses) to be loaded into the segment registers.
    instructions : *lds, les, lfs, lgs, lss.*

In the "Intel 64 and IA-32 Architectures Software Developer's Manual", categorization of *general-purpose instruction* is based on method of use and function [20]. This instruction category can increase resilience during extraction of software birthmark by being divided into wide range but it could cause false-positive in similarity. Accordingly, it is difficult to apply it in the method proposed in this paper.

Pin provides the *INS_Category* function of converting instruction code into instruction category. As shown in Table II, instruction category of Pin has been divided more specifically than the *general-purpose instruction* of the Intel's manual. What we want is to extract birthmark through the level between the *general-purpose instruction* of the Intel's manual and the instruction category of Pin. That's why we are categorizing instruction categorization of Pin once again in middle level to simultaneously satisfy resilience and similarity.

C. Regrouping of Instruction Categories

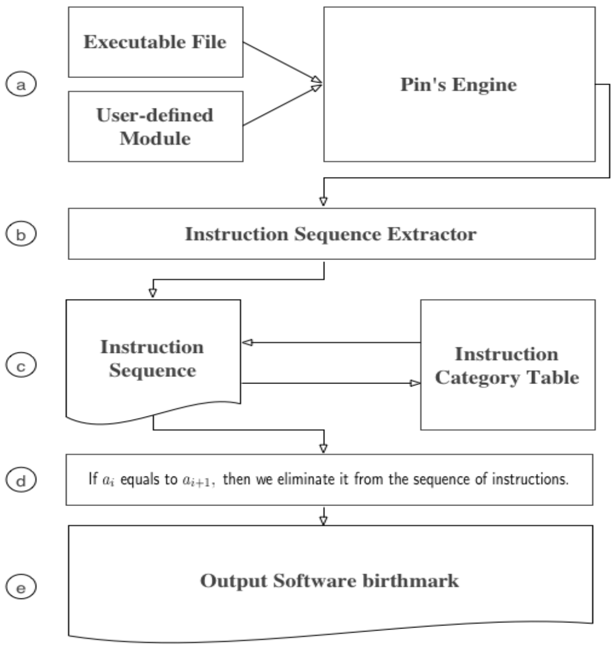
In Table III, instruction categorization of Pin that has been categorized relatively in specific was categorized once again among categories with similar function. Category with clear property such as DATAXFER, INTERRUPT and STRING was applied without any change. In addition, category of POP and PUSH will also be applied without any change since it has important meaning of reading or writing value to stack and it is a collection of frequently used commands. However, COND_BR and UNCOND_BR will be merged as BRANCH since they are same branch processing with difference in existence of condition. In addition, CALL or RET category is a collection of instructions that have significant effects on program flow in spite of very small number of instructions. Accordingly, this will also be applied without any change.

Such instruction categorization will consists of table in source code for experiment. In addition, instruction set such as MMX, SSE and X87 FPU was not dealt with in this paper since it only applies to general-purpose instruction. In the experiment, instruction category in addition to *general purpose instruction* will be applied without any change in the instruction categorization of Pin.

TABLE III: REGROUPING OF INSTRUCTIONS CATEGORIES

PIN	PROPOSE
DATAXFER	DATAXFER
POP	POP
PUSH	PUSH
BINARY	ARITHMETIC
DECIMAL	
LOGICAL	LOGICAL
ROTATE	ROTSFT
SHIFT	
BITBYTE	BITBYTE
COND_BR	BRANCH
UNCOND_BR	
CALL	CALL
RET	RET
INTERRUPT	INTERRUPT
STRINGOP	STRING
IO	IO
IOSTRINGOP	
MISC	MISC
NOP	NOP
FLAGGOP	FLAGSEG
SEGOP	

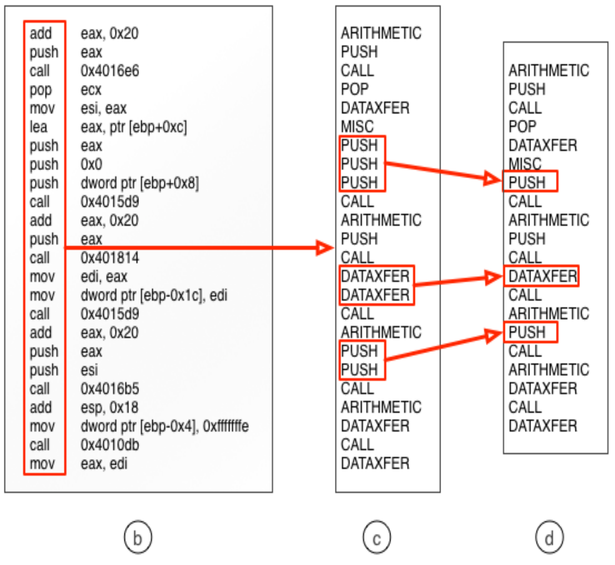
D. Implementation



a
Executable File
User-defined Module
Pin's Engine
b
Instruction Sequence Extractor
c
Instruction Sequence
Instruction Category Table
d
If a_i equals to a_{i+1}, then we eliminate it from the sequence of instructions.
e
Output Software birthmark

Fig. 1. The architecture of dynamic birthmark system.

Fig. 1 shows the process of creating software birthmark through the instruction categorization proposed in this paper. The creation process is divided into 5 steps. In addition, extraction of instruction sequence in step b will be conducted through the instrumentation APIs provided by Pin in the user-defined module in step ***a***. As discussed in Verse 3.1, we will approach through the two methods of routine function (*RTN_AddInstrumentFunction*) and trace function (*TRACE_AddInstrumentFunction*). Fig. 2 shows the change in the instruction sequence created in each step of *b*, *c*, *and d*.



```
add     eax, 0x20
push    eax
call    0x4016e6
pop     ecx
mov     esi, eax
lea     eax, ptr [ebp+0xc]
push    eax
push    0x0
push    dword ptr [ebp+0x8]
call    0x4015d9
add     eax, 0x20
push    eax
call    0x401814
mov     edi, eax
mov     dword ptr [ebp-0x1c], edi
call    0x4015d9
add     eax, 0x20
push    eax
push    esi
call    0x4016b5
add     esp, 0x18
mov     dword ptr [ebp-0x4], 0xfffffffe
call    0x4010db
mov     eax, edi
```

```
ARITHMETIC
PUSH
CALL
POP
DATAXFER
MISC
PUSH
PUSH
PUSH
CALL
ARITHMETIC
PUSH
CALL
DATAXFER
DATAXFER
CALL
ARITHMETIC
PUSH
PUSH
CALL
ARITHMETIC
DATAXFER
CALL
DATAXFER
```

```
ARITHMETIC
PUSH
CALL
POP
DATAXFER
MISC
PUSH
CALL
ARITHMETIC
PUSH
CALL
DATAXFER
CALL
ARITHMETIC
PUSH
CALL
ARITHMETIC
DATAXFER
CALL
DATAXFER
```

Fig. 2. Column of text.

- Start Step

Program to extract and module developed through instrumentation API are entered into PIN's engine for execution.

- Step of extracting instruction sequence

One of routine method and trace method is selected to extract instruction sequence.

- Step of replacement through instruction categorization table

Original instruction is replaced through instruction categorization table.

- Step of removing consecutive duplication

When an instruction of instruction sequence occurs consecutively for more than twice, duplication is removed by removing it in instruction sequence.

- Step of creating birthmark

Result of removing consecutive duplication with instruction categorization is created as birthmark.

IV. Experiment

For the purpose of extracting dynamic birthmark, we discussed earlier about a method of simultaneously satisfying resilience and similarity by reflecting unique characteristic while not damaging its meaning from the sequence of instruction through the categorization of instructions. In this section, this will be verified through two types of experiments. In the first experiment, we experimented the decrease in volume of the extraction result of birthmark. Massive collection of the instructions of program causes much overhead in comparing similarity.

TABLE IV: Experiment of Software Birthmark Capacity

Program	Version	File Size	Routine		Trace	
			All Ins	Categorization Ins	All Ins	Categorization Ins
7z.exe	9.20	160kb	29,832kb	45kb	848kb	31kb
DOSBox.exe	0.74	3640kb	77,259kb	80kb	2,820kb	156kb
Javac.exe	1.7	16kb	47,439kb	67kb	7,415kb	275kb
PEview.exe	0.9	66kb	77,515kb	94kb	10,680kb	417kb
qemu.exe	0.10	1,413kb	43,210kb	54kb	1,147kb	43b

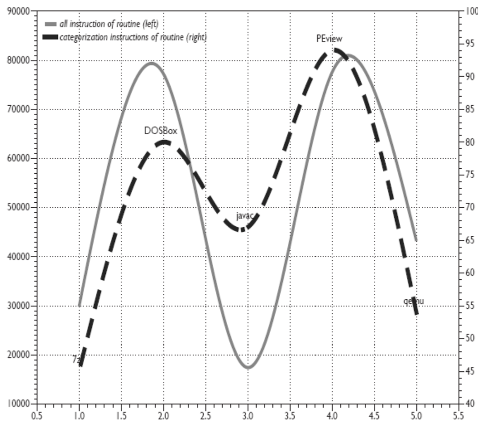


Fig. 3. Compared the capacity of birthmark through Routine function.

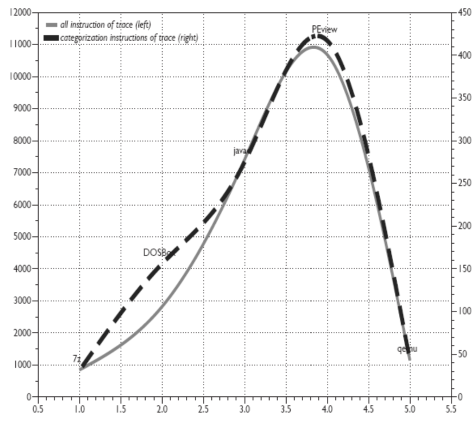
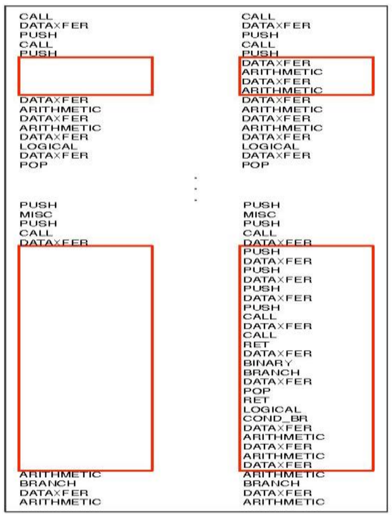


Fig. 4. Compared the capacity of birthmark through trace function.

As shown in Table IV, we conducting benchmarking with five execution files of different capacities. We were able to reduce birthmark volume up to in the average of 95% through our proposal, as shown in the experiment result. As the volume decreased, it could be effective in measuring similarity and the meaning of instruction sequence was not damaged. In Fig. 3 and Fig. 4, we measured to compare our proposed birthmark approach and all over instruction sequences with both *RTN_InstruemntFunction* and *TRACE_ InstrumentFunction*. It is worth stating at this point that a excutable program sequences is kept by semantically control flow, even though it have removed for instructions with consecutive duplication. In the second experiment, we developed a simple program and the program source code was slightly modified to compare the similarity of each program that has been built.



```
CALL                  CALL
DATAXFER              DATAXFER
PUSH                  PUSH
CALL                  CALL
PUSH                  PUSH
                      DATAXFER
                      ARITHMETIC
                      DATAXFER
                      ARITHMETIC
DATAXFER              DATAXFER
ARITHMETIC            ARITHMETIC
DATAXFER              DATAXFER
ARITHMETIC            ARITHMETIC
DATAXFER              DATAXFER
LOGICAL               LOGICAL
DATAXFER              DATAXFER
POP                   POP
           .
           .
           .
PUSH                  PUSH
MISC                  MISC
PUSH                  PUSH
CALL                  CALL
DATAXFER              DATAXFER
                      PUSH
                      DATAXFER
                      PUSH
                      DATAXFER
                      PUSH
                      DATAXFER
                      PUSH
                      CALL
                      DATAXFER
                      CALL
                      RET
                      DATAXFER
                      BINARY
                      BRANCH
                      DATAXFER
                      POP
                      RET
                      LOGICAL
                      COND_BR
                      DATAXFER
                      ARITHMETIC
                      DATAXFER
                      ARITHMETIC
                      DATAXFER
ARITHMETIC            ARITHMETIC
BRANCH                BRANCH
DATAXFER              DATAXFER
ARITHMETIC            ARITHMETIC
```

Fig. 5. Similarities between simple programs.

As shown in Fig. 5, additionally modified area in its meaning can be easily found even consecutive duplication is removed through the categorization of instruction. Through this experiment, we can estimate that more accurate similarity can be measured than the syntax frequency measurement of a certain element that already exists also in a more expanded program or an entirely different program.

V. Conclusion

Software birthmark refers to a technology of extracting unique characteristics of program. This can be used in detecting software theft or malware by comparing the similarity between programs. Software birthmark can be divided into static birthmark and dynamic birthmark according to extraction method. Since information is extracted in executed state of program in dynamic method, relatively more accurate unique information can be extracted compared to static method. However, birthmark was extracted in related works through elements that are insufficient in simultaneously satisfying the resilience and uniqueness of program. In this paper, birthmark extraction method through instruction categorization was proposed to simultaneously satisfy resilience and uniqueness. This proposal increased efficiency in measuring similarity by reducing the massive amounts of instruction sequence up to in the average of 95%. In addition, it was verified that instruction sequence does not become damaged in its meaning in spite of categorization through the comparison of birthmark between programs. Software birthmark is an advancing technology that is being widely used in the area of digital forensic such as code theft. It is expected that program uniqueness and resilience can be increased by extracting birthmark through the method proposed in this paper.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (2014R1A1A2002775).

REFERENCES

[1] H. Tamada, K. Okamoto, M. Nakamura, and A. Monden, "Dynamic software birthmarks to detect the theft of windows applications," presented at International Symposium on Future Software Technology, Oct. 004.

[2] B. Lu, F. Liu, X. Ge, B. Liu, and X. Luo, "A software birthmark based on dynamic opcode n-gram," in *Proc. International Conference on Semantic Computing (ICSC 2007)*, Sep. 2007, pp. 37–44.

[3] H.-I. L. S. Choi, H. Park, and T. Han, "A static api birthmark for windows binary executables," *The Journal of Systems and Software*, vol. 82, no. 5, pp. 862–873, May 2009.

[4] G. Myles and C. Collberg, "Software theft detection through program identification," *The University of Arizona*, 2006.

[5] S. Choi, H. Park, H. I. Lim, and T. Han, "A static birthmark of binary executables based on API call structure," *Advances in Computer Science–ASIAN 2007*, pp. 2–16, 2007.

[6] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in *Proc. the 10th ACM conference on Computer and communications security*, Oct. 2003, pp. 290–299.

[7] J. Kim, D. Lee, W. Jeon, Y. Lee, and D. Won, "Security analysis and improvements of Two-Factor mutual authentication with key agreement in wireless sensor network," *Sensors*, vol. 14, no. 4, pp. 6443-6462, April 2014.

[8] Y. Choi, D. Lee, J. Kim, J. Jung, J. Nam, and D. Won, "Security enhanced user authentication protocol for wireless sensor networks using Elliptic curves," *Sensors*, vol. 14, no. 6, pp. 10081-10106, Jun. 2014.

[9] J. Nam, K.–K. R. Choo, J. Kim, H.-K. Kang, J. Kim, J. Paik, and D. Won, "Method for extracting valuable common structures from heterogeneous rooted and labeled tree data," *Journal of Information Science and Engineering*, vol. 30, no. 3, pp. 787-817, May, 2014.

[10] W. Jeon, J. Kim, J. Nam, Y. Lee, and D. Won, "An enhanced secure authentication scheme with anonymity for wirelessenvironment," *IEICE Transactions on Communications*, vol. E95-B, no. 7, pp. 2505-2508, July 2012.

[11] C. K. Luk, R. Cohn, R. Muth, H. Patil, and A. Klauser, "Pin: building customized program analysis tools with dynamic instrumentation," *Acm Sigplan Notices*, vol. 40, no. 6, pp. 190–200, June 2005.

[12] C. S. Collberg and C. Thomborson, "Watermarking, tamper-proofing, and obfuscation-tools for software protection," *IEEE Transaction on software engineering*, vol. 28, no. 8, pp. 735–746, 2002.

[13] D. Schuler, V. Dallmeier, and C. Lindig, "A dynamic birthmark for java," in *Proc. the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*, pp. 274–283, Nov. 2007.

[14] G. Myles and C. Collberg, "Detecting software theft via whole program path birthmarks," *Information Security*, Berlin, Heidelberg: Springer, pp. 404–415, 2004.

[15] T. H. S. Choi and W. Cho, "A functional unit dynamic api birthmark for windows programs code theft detection," *Journal of KIISE*, vol. 36, no. 9, pp. 767–776, Sep. 2009.

[16] L. Ma, Y. Wang, F. Liu, and L. Chen, "Instruction-Words based software birthmark," in *Proc. 2012 Fourth International Conference on Multimedia Information Networking and Security (MINES)*, Nov. 2012, pp. 909–912.

[17] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," *ACM SIGPLAN Notices*, vol. 42, no. 6, pp. 89–100, 2007.

[18] S. Berkowits. PIN - A Dynamic Binary Instrumentation Tool. [Online]. Available: http://www.software.intel.com/enus/articles/pin-a-dynamic-binary-instrumentation-tool

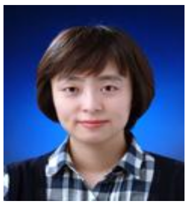
[19] Intel Corporation. (2012). Pin 2.11 User Manual. [Online]. Available: http://rogue.colorado.edu /Pin

[20] Intel Coporation, *64 and IA-32 Intel Architecture Software Developer's Manual Combined*, June 2013.

Donghoon Lee received the B.S. degree of computer science from National Institute for Lifelong Education (NILE), Korea, in 2009 and the M.S. degree in electrical and computer engineering from Sungkyunkwan University, Korea, in 2011. He is currently undertaking a Ph.D. course on electrical and computer engineering in Sungkyunkwan University. His current research interests include malware detection, cryptography, software engineering and information security.

Younsung Choi received the B.S. degree in electrical and computer engineering from Sungkyunkwan University, Korea, in 2006 and the M.S. degree in electrical and computer engineering from Sungkyunkwan University, Korea, in 2007. He is currently undertaking a Ph.D. course on electrical and computer engineering in Sungkyunkwan University. His current research interests include digital forensic, cyber-crime, cryptography, authentication protocol, and mobile security.

Jaewook Jung received the B.S. degree in electrical and computer engineering from Korea Aerospace University, Korea, in 2010 and the M.S. degree in electrical and computer engineering from Sungkyunkwan University, Korea, in 2012. He is currently undertaking a Ph.D. course on electrical and computer engineering in Sungkyunkwan University. His current research interests include cryptography, forensic, authentication protocol, and mobile security.



Jiye Kim received the B.S. degree in Information Engineering from Sungkyunkwan University, Korea, in 1999 and the M.S. degree in Computer Science Education from Ewha Womans University, Korea, in 2007. He is currently undertaking a Ph.D. course on Electrical and Computer Engineering in Sungkyunkwan University. His current research interests include cryptography, forensic, authentication protocol, and information security.

Dongho Won received his B.E., M.E., and Ph.D. from Sungkyunkwan University in 1976, 1978, and 1988, respectively. After working at ETRI (Electronics & Telecommunications Research Institute) from 1978 to 1980, he joined Sungkyunkwan University in 1982, where he is currently a professor of the School of Information and Communication Engineering. In the year 2002, he served as the president of KIISC (Korea Institute of Information Security & Cryptology). He was the Program Committee Chairman of the 8th International Conference on Information Security and Cryptology (ICISC 2005). His current research interests include cryptology and information security.