# Software Birthmark Method Using Combined Structure–Based and API–Based

Donghoon Lee, Dongwoo Kang, Jiye Kim, and Dongho Won*
College of Information and Communication Engineering, Sungkyunkwan University, Suwon, 440-746, Korea.

* Corresponding author. Tel.: +82-31-290-7213; email: {dhlee, dwknag, jykim, dhwon}@security.re.kr
Manuscript submitted October 10, 2016; accepted December 10, 2016.
doi: 10.17706/jsw.12.2.138-144

**Abstract:** The software birthmarking technique has traditionally been studies in fields such as software piracy, code theft, and copyright infringement. In this paper, we propose a static software birthmark technique that is combined by the structure–based and API–based. Our proposed software birthmark technique is based on procedures that contain the call sequence, including the interprocedural and APIs in the distributed softwares with native code. The procedure birthmark is translated to the ordered tree by call-chain of interprocedural analysis. Finally, the software birthmark generates a list of *pq*-gram with the ordered trees translated from each procedure. Our experiment performed with variant malwares. These malware are shown that can be distinguished from one another in our method.

**Key words:** Software birthmark, software similarity, static analysis, tree edit distance, code theft.

## 1. Introduction

A software birthmark is a technology that reflects the characteristics that are inherent to each software application, and it has conventionally been studied for detection of software piracy, code theft, and copyright infringement. According to the 4th Annual State of Application Security Report published by ARXAN in 2015, approximately 77.9% of illegally shared media worldwide is software [1]. That software alliance analysis and IDC findings also reported that 45% of illegal software distributions are performed through online websites or P2P networks [2].

Software vendors have studied and applied such technologies such as *software watermarking* [3] [4], *tamper-proofing* , *obfuscating* [5] [6], and *software birthmark* [7]-[10] to protect their intellectual property. Among these, software birthmark technique represents a technology that reflects the inherent features of each software program. Therefore, such technologies are widely used in many recent applications, including digital forensics, malicious code detection, and detection of software copyright infringement and code theft [11].

Software birthmarks have also been proposed in the graph structure of a program. Each function (alternatively, procedures on native code) of a program can be expressed as the dependence among statements in a function, the inheritance relationship between classes (such as acyclic graphs), and the control flow. Accordingly, a birthmark can be generated as an expression of a program graph [12]. Myles and Collberg proposed for Java applications a dynamic birthmark called the whole program path (WPP) [7]. To extract the WPP birthmark, dynamic traces of a program are compressed into a directed acyclic graph and then collected. However, comparing two graphs with millions of nodes may prove prohibitively expensive;

moreover, it is unclear how this birthmark would perform on substantial traces of real programs [12].

Several birthmarks exist that are based on the way a program uses standard libraries or system calls (henceforth collectively referred to as APIs); such a birthmark is both unique to that program and difficult for an attacker to forge [12]. We classify these birthmarks as API–based ones. Tamada et al. presented three algorithms for collecting birthmarks. These algorithms compute the birthmark from the sequence of method calls within a class, the inheritance path from the root class to a given class, and the types that a class employs, respectively, [13] [14]. In addition, Park et al. proposed a static API–call–based birthmark for software theft detection of Java applications [15]. Choi et al. additionally presented a static API birthmark for Windows execution files using a set of API calls identified as being static by a disassembler [16]. In addition to the above static birthmark–generation techniques, several dynamic API–based birthmarks have been proposed. Tamada et al. suggested a method of tracing the API calls of programs that are executed by particular input values [10]. Schuler et al. proposed a method of combining $k$–gram–based birthmarks and API-based birthmarks [9]. These researchers constructed a set of $k$–grams for API call sequences and proposed dynamic $k$–gram API–based birthmarking using an API call sequence that is well known to the program being executed with particular input values.

In this paper, we propose a static software birthmark technique that is combined by the structure–based and API–based. The remainder of this paper is organized as follows. In Section. 2, we explain relevant knowledge to elucidate our proposed method. Next, Section. 3 we provide a detailed description of our proposed method. Our experiments describe in Section. 4. Section. 5 presents our conclusion and future work.

## 2. Preliminaries

### 2.1. Software Birthmark

The software similarity problem has conventionally focused on code theft detection. It is used to determine if program $\mathcal{P}$ is a copy or derivative of program $\mathcal{Q}$. It is an extension of the definition in [10] and [17]. A workflow is shown in Fig. 1.

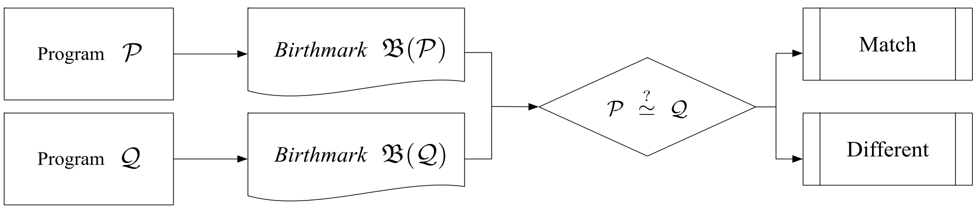



Fig. 1 Software similarity problem.

**Definition 1: (Software Birthmark)** Let □ denote all sets of a program, where program $\mathcal{P}$ is given as $\mathcal{P} \in$ □. Let $\mathfrak{B}_p$ denote a function capable of extracting a set of program characteristics. If the following conditions are satisfied, the birthmark $\mathfrak{B}_p(\mathcal{P})$ of program $\mathcal{P}$ can be defined.

- $\mathfrak{B}_p(\mathcal{P})$ is obtained only from $\mathcal{P}$ itself.
- Program $\mathcal{Q} \in$ □ is a copy of $\mathcal{P} \longrightarrow \mathfrak{B}_p(\mathcal{P}) = \mathfrak{B}_p(\mathcal{Q})$.

As shown in Def 1, the software birthmark reflects the innate traits extracted from program $\mathcal{P}$ itself. Such a software birthmark is a technology designed to measure the similarity of two programs. If similarities of birthmarks extracted from two programs are matched, then the two programs can be considered identical or copied.

In order to evaluate the effectiveness of software birthmarks, researchers usually consider two properties: credibility and resilience [18].

**Property 1: (Resilience)** Let us first assume there are two programs or program computers $\mathcal{P}, \mathcal{Q} \in$ ☐. Then, let us say that $\mathfrak{B}_p(\mathcal{P}) \longrightarrow \alpha$ and $\mathfrak{B}_p(\mathcal{P}) \longrightarrow \beta$ are the birthmark values extracted from programs $\mathcal{P}$ and $\mathcal{Q}$. Let $\boldsymbol{Sim}_p(\alpha, \beta) \longrightarrow [0, 1]$ be a function that measures program similarity; the threshold is given as $0 < \varepsilon < 1$. If $\mathcal{P}$ and $\mathcal{Q}$ are similar to each other and $\boldsymbol{Sim}_p(\alpha, \beta) > 1 - \varepsilon$, then the birthmarking system is called resilient.

**Property 2: (Credibility)** Let $\mathcal{P}$ and $\mathcal{Q}$ denote independantly written programs. If the birthmarking system can distinguish between the two programs, it is deemed reliable and can be defined as follows:

$$\boldsymbol{Sim}_p(\alpha, \beta) \leq \varepsilon \tag{1}$$

Properties 1 and 2 define the basic properties in measuring similarity between two birthmarks. Credibility defines the property in which comparison values of programs from the birthmarking system can be clearly sorted.

## 2.2. $pq$–Gram Distance

The $pq$–gram distance is a tree distance function for ordered trees, in which the main idea is to break down trees into constant-sized fragments called $pq$–grams, which represent both tree structure and content [19]. Next, given two trees, [19] measures their similarity by comparing their $pq$–gram multisets using the formula in the following example.

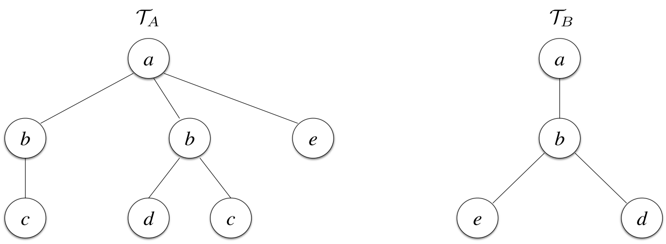



Fig. 2. Two trees $\mathcal{T}_A$ and $\mathcal{T}_B$.

**Example 1. ($pq$–gram Distance)** Consider the trees $\mathcal{T}_A$, $\mathcal{T}_B$ shown in Fig. 2. Their corresponding $pq$–gram multisets (for $p = 2, q = 1$) are:

$$\varphi^t_{2,1}(\mathcal{T}_A) = \{(*, a; b), (*, a; b), (*\, a; e), (a, b; c), (a, b; d), (a, b; c), (a, e;*), (b, c;*), (b, d;*), (b, c;*)\}$$
$$\varphi^t_{2,1}(\mathcal{T}_B) = \{(*, a; b), (a, b; e), (a, b; d), (b, e;*), (b, d;*)\}$$

Using the following normalized $pq$–gram distance function $\boldsymbol{dist}^{p,q}_{norm}$ [19] to calculate the distance between $\mathcal{T}_A$, $\mathcal{T}_B$,

$$\boldsymbol{dist}^{p,q}_{norm}(\mathcal{T}_A, \mathcal{T}_B) = \frac{|\varphi^t_{2,1}(\mathcal{T}_A) \cup \varphi^t_{2,1}(\mathcal{T}_B)| - 2|\varphi^t_{2,1}(\mathcal{T}_A) \cap \varphi^t_{2,1}(\mathcal{T}_B)|}{|\varphi^t_{2,1}(\mathcal{T}_A) \cup \varphi^t_{2,1}(\mathcal{T}_B)| - |\varphi^t_{2,1}(\mathcal{T}_A) \cap \varphi^t_{2,1}(\mathcal{T}_B)|} \tag{2}$$

we obtain a distance value of $\frac{15-2\cdot 3}{15-3} = 0.75$. Note that this distance formula is normalized to the range [0, 1]. A non–normalized version has also been defined in [19].

The $pq$–gram distance is $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space, where $n$ is the number of tree nodes. As shown in [19], the $pq$–gram distance can be used as a lower bound on fanout weighted tree edit distance, which is a variant of tree edit distance that is defined in the same paper.

In this paper, our proposed method uses a $pq$–gram distance to measure the similarity between two procedures that was builded to the ordered trees with the APIs and interprocedural calls, and its details explain Section 3.

## 3. Proposed Method

### 3.1. Feature Extraction

Our proposed software birthmark technique is based on procedures that contain the call sequence, including the interprocedural and APIs in the distributed softwares with native code. In other words, these features are extracted from a single software and are described as follows:

- A set of procedures that call sequence, including the APIs and interprocedureal.
- The names or label in each procedure.

With that features, we generate each procedure birthmark. Next, the procedure birthmark is translated to the ordered tree by call-chain of interprocedural analysis. Finally, the software birthmark function $\mathfrak{B}_p$ generates a list of $\varphi^t_{p,q}$ with the ordered trees translated from each procedure.

### 3.2. Software Birthmark Generation

In our method, the software birthmark is based on call sequence of the segmented procedures in a single software. Then, the procedure birthmark is defined as below.

**Definition 2: (Procedure Birthmark)** Given a program $\mathcal{P} \in$ ?, let us assume that is a universal set of procedures $\mathcal{P} = \{f_1, f_2, f_3, \ldots, f_{n-1}, f_n\}$, where $n$ is the size of the procedures in a program $\mathcal{P}$. Also, let us assume that $f_i$ is specific $i$-th procedure of the sequence in $\mathcal{P}$, where $i$ has $0 < i \leq n$. Then, a procedure birthmark function $\mathfrak{B}_f$ is defined as following:

$$\mathfrak{B}_f(f_i) = \{name;\ c_1, c_2, \ldots, c_m \mid m \geq 0\} \tag{3}$$

where $name$ is a key value of $f_i$ (such as sub_4010F2, sub_4B70FF), and $c_i$ is the call sequence, including the call instruction of interprocedural or APIs in a procedure $f_i$. If $m$ is 0, then $f_i$ can only hold the $name$.

Next, the software birthmak function $\mathfrak{B}_p$ can be defined as following:

$$\mathfrak{B}_p(\mathcal{P}) = \left\{ \bigcup_{i=1}^{n} \varphi^t_{p,q}(\,\mathrm{T}(\,\mathfrak{B}_f(f_i)\,) \rightarrow \mathcal{T}_i\,) \right\}, \tag{4}$$

where T is a function that converts to tree $\mathcal{T}_i$ from $\mathfrak{B}_f(f_i)$, and $\mathcal{T}_i$ is translated by $\varphi^t_{p,q}$ described in Example. 1. Consequentially, the software birthmark function $\mathfrak{B}_p$ is the set of $\varphi^t_{p,q}$ for $\mathrm{T}(\,\mathfrak{B}_f(f_i)\,) \rightarrow \mathcal{T}_i$ in a software $\mathcal{P}$.

Our detailed example of $\mathrm{T}(\,\mathfrak{B}_f(f_{\mathrm{sub\_4054A0}})\,)$ are presented in Fig. 3.

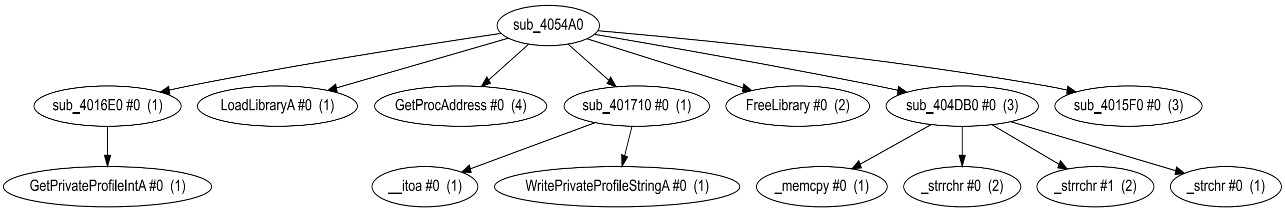



Fig. 3 $\mathcal{T}_{\text{sub\_4054A0}}$ translated by $\mathrm{T}(\mathfrak{B}_f(f_{\text{sub\_4054A0}}))$

### 3.3. Measuring Similarity with Software Birthmark

Our goal is to measure similarity between two softwares. To achieve this, the list of birthmarking procedures is measured by $pq$–gram distance function $\boldsymbol{dist}_{norm}^{p,q}$.

**Definition 3: (Software Similarity)** Let us assume there are two software $\mathcal{P}, \mathcal{Q} \in$ ▯. Also, let us say that $\mathfrak{B}_f(f_i \in \mathcal{P}) \longrightarrow \alpha_i$ and $\mathfrak{B}_f(f_i \in \mathcal{Q}) \longrightarrow \beta_i$ are birthmark values extracted by procedure birthmark function $\mathfrak{B}_f$. Then, software similarity $\boldsymbol{Sim}_p(\mathcal{P}, \mathcal{Q})$ is defined as:

$$\boldsymbol{Sim}_p(\mathcal{P}, \mathcal{Q}) = \frac{2 \cdot \sum_{i=1}^{n} \boldsymbol{max}\left(\bigcup_{j=1}^{k} \boldsymbol{dist}_{norm}^{p,q}(\alpha_i, \beta_j)\right)}{n + k} \tag{5}$$

where $n$ and $k$ represent the number of procedures that contain APIs and interprocedural calls for two software $\mathcal{P}$ and $\mathcal{Q}$, respectively. $\alpha_i$ and $\beta_j$ are the values generated through $\mathfrak{B}_f$ for each procedure extracted from the two software $\mathcal{P}$ and $\mathcal{Q}$.

## 4. Experiment

Our experimental environment was comprised of the 32–bit Windows 7 operating system, an Intel Core i7 2.6Ghz processor, and 16GB of RAM. The system was embodied by Python, and the packages of the pefile and distorm3 were used. Additionally, IDA pro 6.1 of Hex–Rays was used for the verification of the disassembly.

Our experiment was performed on malware or variant malware (as shown in Table I.).

Table 1. Similarity Measure for Several Malware and Its Variant, Using our Proposed Method

| $p$=2, $q$=3 | Roron.25 | Roron.31 | Roron.41 | Wuke.a | Wuke.b | Wuke.f | Ramm.i | Ramm.j | Ramm.m |
|---|---|---|---|---|---|---|---|---|---|
| Roron.25 | 1.00 | 0.73 | 0.76 | 0.46 | 0.46 | 0.12 | 0.71 | 0.50 | 0.65 |
| Roron.31 | 0.73 | 1.00 | 0.69 | 0.34 | 0.34 | 0.10 | 0.42 | 0.25 | 0.36 |
| Roron.41 | 0.76 | 0.69 | 1.00 | 0.37 | 0.36 | 0.13 | 0.43 | 0.25 | 0.37 |
| Wuke.a | 0.46 | 0.34 | 0.37 | 1.00 | 0.98 | 0.87 | 0.79 | 0.62 | 0.74 |
| Wuke.b | 0.46 | 0.34 | 0.36 | 0.98 | 1.00 | 0.79 | 0.79 | 0.62 | 0.74 |
| Wuke.f | 0.12 | 0.10 | 0.13 | 0.87 | 0.79 | 1.00 | 0.75 | 0.69 | 0.71 |
| Ramm.i | 0.71 | 0.42 | 0.43 | 0.79 | 0.79 | 0.75 | 1.00 | 0.84 | 0.95 |
| Ramm.j | 0.50 | 0.25 | 0.25 | 0.62 | 0.62 | 0.69 | 0.84 | 1.00 | 0.83 |
| Ramm.m | 0.65 | 0.36 | 0.37 | 0.74 | 0.74 | 0.71 | 0.91 | 0.83 | 1.00 |

In Table I, the results shows that our method that our method is an possibility in which API-based and structure-based were combined to the ordered tree. Especially, Wuke and Roron are certainly distinguished the type of each other, and the same families was similar. In our experiment, the $p$ and $q$ values on the $pq$–gram distance function $\boldsymbol{dist}_{norm}^{p,q}$ was 2 and 3, respectively.

## 5. Conclusion and Future Work

In this paper, we proposed a static software birthmark technique that is combined by the structure–based and API–based. Our proposed software birthmark technique is based on procedures that contain the call sequence, including the interprocedural and APIs in the distributed softwares with native code. The procedure birthmark is translated to the ordered tree by call-chain of interprocedural analysis. Finally, the software birthmark generates a list of *pq*-gram with the ordered trees translated from each procedure. Our experiment was performed on malwares or variant malwares. The results showed that our method is an possibility in which API-based and structure-based were combined to the ordered tree.

Our future work will address unknown or variant malware detection as an extension of this proposed birthmark technique. Also, we need to determine threshold $\varepsilon$ through more experiments with the application and malwares.

## Acknowledgment

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funcded by the Ministry of Science, ICT, and Future Planning (2014R1A 1A2002775).

## References

[1] ARXAN. (2015). *A Look Inside the Universe of Pirated Software and Digital Assets.* (Tech. report). 4th Annual State of Application Security Report.

[2] Gantz, J. F., *et al*. (2013), *The Dangerous World of Counterfeit and Pirated Software* (Tech. report). IDC White Paper.

[3] Collberg, C., & Thomborson, C. (1999). Software watermarking: Models and dynamic embeddings. *Proceedings of the 26th ACM SIGPLAN–SIGACT Symposium on Principles of Programming Language* (pp. 311–324). ACM.

[4] Zhu, W., Thomborson, C., & Wang, F. Y. (2005). A survey of software watermarking. *Proceedings of the Lecture Notes in Computer Science: Vol. 3495. IEEE International Conference on Intelligence and Security Informatics* (pp. 454–458).

[5] Collberg, C., Thomborson, C., & Low, D. (1997). *A Taxonomy of Obfuscating Transformations*. Department of Computer Science, The University of Auckland, New Zealand.

[6] Collberg, C. S., & Thomborson, C. (2002). Watermarking, tamper–proofing, and obfuscation–tools for software protection. *IEEE Transactions on software engineering*, *28(8)*, 735–746.

[7] Myles, G., & Collberg, C. (2004, September). Detecting software theft via whole program path birthmarks. *Proceedings of International Conference on Information Security* (pp. 404–415). Springer Berlin Heidelberg.

[8] Myles, G., & Collberg, C. (2005, March). K–gram based software birthmarks. *Proceedings of the 2005 ACM symposium on Applied computing* (pp. 314–318). ACM.

[9] Schuler, D., Dallmeier, V., & Lindig, C. (2007, November). A dynamic birthmark for java. *Proceedings of the twenty–second IEEE/ACM International Conference on Automated Software Engineering* (pp. 274–283). ACM.

[10] Tamada, H., Okamoto, K., Nakamura, M., Monden, A., & Matsumoto, K. I. (2004, October). Dynamic software birthmarks to detect the theft of windows applications. *Proceedings of the International Symposium on Future Software Technology*.

[11] Garfinkel, S. L. (2010). Digital forensics research: The next 10 years. *Digital Investigation*, *7*, S64–S73.

[12] Collberg, C., & Nagra, J. (2009). *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison–Wesley Professional.

[13] Tamada, H., Nakamura, M., Monden, A., & Matsumoto, K. I. (2003). Detecting the theft of programs using birthmarks. *NAIST–IS–TR2003014*.

[14] Tamada, H., Okamoto, K., Nakamura, M., Monden, A., & Ichi, M. K. (2007). Design and evaluation of dynamic software birthmarks based on api calls. *Info. Science Technical Report NAIST–IS–TR2007011, ISSN*, 0919–9527.

[15] Park, H., Choi, S., Lim, H. I., & Han, T. (2008, November). Detecting Java theft based on static API trace birthmark. *International Workshop on Security* (pp. 121–135). Springer Berlin Heidelberg.

[16] Choi, S., Park, H., Lim, H. I., & Han, T. (2009). A static API birthmark for Windows binary executables. *Journal of Systems and Software*, *82*(*5*), 862–873.

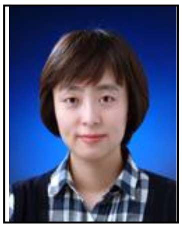
[17] Cesare, S., & Xiang, Y. (2012). Software birthmark similarity. *Software Similarity and Classification* (pp. 63–70). Springer London.

[18] Myles, G. (2006). Software theft detection through program identification.

[19] Augsten, N., Böhlen, M., & Gamper, J. (2010). The *pq*–gram distance between ordered labeled trees. *ACM Transactions on Database Systems (TODS)*, *35(1)*, 4.

**Donghoon Lee** received the M.S. degree in information security engi- neering from Sungkyunkwan University, Korea, in 2011. He is currently undertaking a Ph.D. course on electrical and computer engineering in Sungkyunkwan University. He also worked as a security developer in egloo security and nexon company between 2010 and 2013. His current research interest is in the area of software security, cryptography, authe-ntication protocol, and network security.

**Dongwoo Kang** was born in Seoul, Korea on January 26, 1993. He received the B.S. degree in electrical and computer engineering from Sungkyunkwan University, Korea, in 2015. He is currently pursuing M.S. degree in electrical and computer engineering at Sung-kyunkwan University. His current research interest includes cryptography, malware, and authentication or key management protocols.



**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 Ehwa University, Korea, in 2007. He is currently undertaking a Ph.D. course on electrical and computer engineering in Sungkyunkwan University. His current research interest is in the area of cryptography, authentication protocol, and sensor 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 and Telecommunications Re- search Institute) from 1978 to 1980, he joined Sungkyunkwan University in 1982, where he is currently 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 and Cryptology). He was the program committee chairman of the 8th International Conference on Information Security and Cryptology (ICISC 2005). His research interests are on cryptology and informationsecurity.