B-005

動的バースマークの自動抽出に向けて
～比較ソフトウェアの入力からの実行系列の自動抽出～

Towards the auto extraction for the dynamic software birthmarks with the inputs from the plaintiff software

セルメニョ アレハンドロ† 玉田 春昭† 神崎 雄一郎‡
Cermeño Alejandro Haruaki Tamada Yuichiro Kanzaki

1. はじめに

ソフトウェアの盗用を検出するための技術として，ソフトウェアバースマークが提案されている[1][2]．この技術は主にソフトウェアの実行ファイルから本質的な情報を抽出し，比較することで類似度を算出する技術である．そして，プログラムの静的情報に基づく静的バースマーク[1]と，動的情報に基づく動的バースマーク[3]に分けられる．本質的な情報とは，静的バースマークでは例えば，opcode の *k*-gram[4]などが，動的バースマークでは，オブジェクト同士の参照グラフ[5]などが提案されている．

バースマークの典型的なシナリオでは，オリジナルのソフトウェア（原告）と盗まれたものであるか判定したいソフトウェア（被告）の 2 つのソフトウェアが登場する．次に，原告ソフトウェア，被告ソフトウェアそれぞれからバースマーク情報を抽出し，比較して，類似度を算出する．その類似度をもとに盗用であるか否かを判定する．

静的バースマークは，主にソフトウェアの部品の盗用検出に特化しており，一方の動的バースマークはソフトウェア全体の盗用検出に向いている．また，静的バースマークは，抽出コストは低いが，プログラムの変形に対して脆弱である場合が多い．対する動的バースマークは，抽出コストが高い反面，プログラムの変形に対して堅牢である場合が多い．動的バースマークの抽出は，ソフトウェアを実行する必要がある．そして，一般的にプログラムの実行経路が異なると，全く異なる情報が得られる．そのため，原告，被告の両者が共に似たような実行経路になるよう入力を選択する必要がある．この入力の選択は，人手による試行錯誤が不可欠であるため自動化が不可能である．このことから動的バースマークの抽出コストは高くなっている．

動的バースマークの抽出コストを下げるための取り組みも行われている．横井らは原告ソフトウェアの抽出コストを下げるため，単体テストコードに着目している[6]．原告ソフトウェアは出自がはっきりしているものであり，テストコードはバグを検出するためソフトウェアの一部を動作させる．そのため，テストコード中のテストメソッドを 1 つの入力と考え，ソフトウェアを実行させて動的バースマークを抽出する．また，松田らは原告ソフトウェアへの入力を，シンボリック実行を用いて，入力集合を自動的に取得しようとしている[7]．ただし，シンボリック実行が必ずしも成功するわけではないため，あらゆるソフトウェアに対して有効であるわけではない点に課題がある．

そこで本稿では，より簡易に動的バースマークを抽出する手段を提案する．提案手法では，前提として原告ソフトウェアはよく知っているものとする．そこで，原告ソフトウェアに与える入力をそのまま被告ソフトウェアにも与えるものとする．この場合に得られたバースマークの性能を評価するものである．

過去に Windows の GUI ソフトウェアに対しては，入力なし（実行後，即座に終了させる）の動的バースマークで評価している[3]．この方法が CUI ソフトウェアに対しても有効であるかを評価するものである．

2. 準備

2.1 ソフトウェアバースマーク

ソフトウェアバースマークは玉田らに よって定義されている [1]．その定義を元に岡本らが動的バースマークを次のように定義した [3].

定義 1（ソフトウェアバースマーク） p, q を与えられたプログラムとし, I を p および q に与える入力とする．そしてプログラム p に入力 I を与えた時に，ある方法 f によって得られる実行時情報から抽出した特徴の集合を $B_f(p, I)$とする．このとき，以下の条件を満たすならば，$B_f(p, I)$を p の I における動的バースマークであると言う．

条件 1. $B_f(p, I)$はプログラム p に入力 I を与えることで得られる．

条件 2. q が p のコピーであれば，$B_f(p, I) = B_f(q, I)$

条件 1 は，バースマークは電子透かしのように，後から追加されるような情報ではなく，プログラムに元来備わっている情報であることを表している．条件 2 は，もし，バースマーク $B_f(p, I)$と $B_f(q, I)$が異なっているならば，q は p のコピーではないことを意味する．ただし，入力が異なれば同じプログラムであっても，一般に異なる動的バースマークが得られる（$B_f(p, I) \neq B_f(p, J)$）．

また，バースマークは以下の保存性（Resilience, Preservation），弁別性（Credibility, Distinction）の 2 つの性質を満たすことが望まれる．

性質 1　保存性（Resilience, Preservation） p から任意の等価変換により得られた p'に対して，$B_f(p, I)=B_f(p', I)$を満たす．

性質 2　弁別性（Credibility, Distinction） 同じ処理を行うプログラム p と q が全く独立に実装された場合，$B_f(p, I) \neq B_f(q, I)$ を満たす．

保存性はバースマークが様々な攻撃に対して耐性を持つことが期待されていることを表す．一方で弁別性は，異な

† 京都産業大学　Kyoto Sangyo University.
‡ 熊本高等専門学校　National Institute of Technology (KOSEN), Kumamoto College.

表 1. 対象ソフトウェア

カテゴリ	ソフトウェア名	バージョン	URL
ファイル一覧	ls	GNU coreutils 8.32	https://www.gnu.org/software/coreutils/
	dir	GNU coreutils 8.32	https://www.gnu.org/software/coreutils/
	exa	0.9.0	https://github.com/ogham/exa
HTTP	curl	7.74.0	https://github.com/curl/curl
	wget	1.21	https://www.gnu.org/software/wget/
	httpie	2.2.0	https://github.com/httpie/httpie
SCM	git	2.30.2	https://git-scm.org
	Mercurial	5.6.1	https://www.mercurial-scm.org
	Apache Subversion	1.14.1	https://subversion.apache.org
Clojure インタプリタ	Clojure	1.11.1.1273	https://clojure.org
	Leiningen	2.10.0	https://leiningen.org
	Babashka	1.3.180 (Java)	https://babashka.org
	Babashka	1.3.180 (Native)	https://babashka.org

る開発者が互いのソースコードを参照し合わずに同じような処理を実装した場合，$B_f(p, I) \neq B_f(q, I)$を満たすことが望まれることを示している．もちろん，この２つの性質を完全に満たすバースマークを提案することは困難である．そのため，実用上はユーザの判断により適宜強度を設定する必要がある．

2.2 バースマークの種類

第 2.1 節に示したバースマークの定義に従い，異なる種類のバースマークが提案されている．これらは，実行時に得られる情報の構成方法や着目する情報により異なる．例えば，呼び出しメソッドに着目した手法 [3]，実行時のヒープに着目した 手法 [5] などである．

一方，同じように抽出した情報であっても構成方法が異なれば，異なるバースマーク情報として定義される．例えば，呼び出しメソッドの系列をそのままシーケンスとして扱う EXESEQ バースマークや呼び出しメソッドの系列をベクトルとして扱う EXEFREQ バースマークなどである [3].

これらの着目する情報や構成方法による違いを f というバースマーク抽出方法にまとめるものとする．言い換えれば，バースマーク抽出方法 f がそのままバースマークの種類であると言える．

2.3 バースマークの類似度

多くのバースマーク f で 得られたバースマーク $B_f(p, I)$ と $B_f(q, I)$ の比較に は，$\mathrm{sim}_f(B_f(p, I), B_f(q, I))$ が定義されている．なお， $\mathrm{sim}_f(B_f(p, I), B_f(q, I))$ の値域は [0, 1] である．

類似度が 0 であれば，両プログラムは完全に独立していることを意味し，類似度が 1 であれば，そのプログラムはコピーである疑いが非常に強いことを意味する．ただし，どの程度 1 に近ければコピーであるのかの判定のため，多くの場合閾値 ε が導入されている．もし $B_f(p, I)$ と $B_f(q, I)$ の類似度が閾値 ε より大きい時，p もしくは q のいずれかが他方からコピーされている疑いがあることを意味する．そして，類似度の結果を以下の 3 つのグループに分類する[8]. 上記のように，類似度が ε 以上の場合は，プログラム p と q

$$\mathrm{sim}_f(B_f(p, I), B_f(q, I)) = \begin{cases} \geq \varepsilon & \textit{copy relation} \\ \leq 1-\varepsilon & \textit{no copy relation} \\ \textit{otherwise} & \textit{inconclusive} \end{cases}$$

はコピー関係の疑いが強いことを表し，類似度が $1-\varepsilon$ 以下の場合は，コピー関係にない可能性が高いことを表す．そして，それ以外（$1-\varepsilon < \mathrm{sim}_f(B_f(p, I), B_f(q, I)) < \varepsilon$）の場合は，バースマークではコピー関係を判定できないことを示している．なお，典型的な ε の値は 0.75 である[9].

3. 提案手法

バースマークを用いた典型的なシナリオは２つ考えられる．１つは自身が開発したソフトウェア S（原告ソフトウェア）と似通ったソフトウェア S'（被告ソフトウェア）を見つけた時に，S と S'の類似性を調査するというものである．もう 1 つは大量のソフトウェア（被告ソフトウェア群）の中から S と似通ったソフトウェアを見つけ出すというものである．どちらのシナリオにせよ，S は自身が作成したソフトウェアであるためコードの詳細までわかっていることが多いと考えられる．そのため，S からバースマークを抽出することはバースマーク抽出法さえわかっていれば容易である．

ソフトウェアの静的な構造に着目している静的バースマークはバースマーク抽出法（ツール）が手元に存在すれば，どのようなソフトウェアからでも抽出が容易である．一方で，ソフトウェアの動的情報を抽出する動的バースマークの場合，効果的なバースマークを抽出するには入力の調整が必要になり，内部構造を理解していないソフトウェアであれば一般に抽出が困難である．入力が異なれば，辿る実行経路は異なるものになり，その結果，異なるバースマークが抽出されることになる．そのため，例え盗用されたものであったとしても，与える入力が間違っていれば異なる実行経路を辿る可能性がある．このことから動的バースマークの抽出コストは高いと言われていた．

しかし，世の中の多くのソフトウェアは盗用ではない．また，盗用されたとしても，原告への入力をそのまま被告に与えたとしても実行経路が大きく変わることは少ないと考えられる．そのため，原告に与える入力をそのまま被告に与えることで，抽出コストを下げられると考えられる．

4. 評価実験

4.1 実験概要

本稿では提案手法を弁別性評価と保存性評価の 2 軸で評価する．弁別性評価では，あるカテゴリからソフトウェアをいくつか選択する．それらのソフトウェアに対して，一律に同じ入力を与えたときの動的バースマークを比較する．

表 2. Babashka のバージョン

バージョン	リリース日	ファイル数	総行数	追加行数	削除行数
0.4.0	2021-05-09	297	27,040	N/A	N/A
0.5.0	2021-06-21	326	32,411	5,926	533
0.6.0	2021-09-02	342	34,801	2,866	475
0.7.0	2021-12-10	380	39,153	5,233	903
0.8.0	2022-04-04	525	61,458	23,612	1,135
0.9.162	2022-09-05	561	63,261	2,807	1,077
1.0.164	2022-10-19	568	63,690	589	154
1.3.180	2023-05-28	579	67,670	5,954	1,879

対象のソフトウェアは OSS から選択する．すなわち，盗用ではないため，類似度は低くなることが期待される．

弁別性評価での対象のソフトウェアを表 1 に示す．4 つのカテゴリ（ファイル一覧，HTTP，SCM，Clojure インタプリタ）からそれぞれ 3 つのソフトウェアを選択している．表 1 にはカテゴリ，ソフトウェア名，利用したバージョン，そのソフトウェアの URL を示している．また，Clojure インタプリタの一つである Babashka は Java で実装されており，リリースに Java のバイナリ形式である jar 以外にも，GraalVM を用いたネイティブコードも提供している[10]．このネイティブコードも，バースマークの文脈では，元のソフトウェアの隠すための方策と扱える．以降，このネイティブコードを BBNative と呼ぶ．

次の保存性評価では，ある 1 つのソフトウェアの異なるバージョンを集め，それぞれに対して同じ入力を与える．バージョン違いは，本来は盗用ではないが，以前のバージョンを元に拡張したものであり，バースマークの文脈においては盗用であると見做せる．

保存性評価の対象ソフトウェアは，表 1 中の Clojure のインタープリタである Babashka の複数のバージョンを利用する．利用したバージョンを表 2 に示す．表 2 でのファイル数，総行数列は当該バージョンに存在するファイル数とそれらの総行数を表す（`tokei`[1]を用いて計測した）．また，追加行数，削除行数列は，前バージョンから当該バージョンに至るまでに追加，削除された行数を表している（`git diff --stat` を用いて計測した）．

いずれの評価においても，`strace` コマンド[2]を用いて，ソフトウェアを起動し，終了するまでのシステムコールを記録する．記録されたものからシステムコールの名前のみを取り出す．そのリストの 6-gram の頻度を当該コマンドのバースマークとして扱う[4]．バースマークの類似度は，頻度をベクトルとして扱い，コサイン類似度を用いる[3]．

実行環境は `openjdk:17-slim-bullseye` をベースに評価対象ソフトウェアをインストールした Docker コンテナを用意した[3]．ホスト PC は MacBook Pro M2, 24GB RAM, macOS Ventura 13.4，Docker Desktop 4.19.0 である．

4.2 実験 1 弁別性評価

ここでは，個別に実装されたソフトウェアがバースマークにより盗用ではないと判定できるかを確認する．対象のソフトウェアに対して，入力なし（コマンドラインに何も引数を与えず実行する），ヘルプ（`--help` をコマンドの後ろにつけて実行する）の 2 つを実行し，システムコールを記録した．また，カテゴリごとに以下のように特有の動作を行い，同様にシステムコールを記録した．以下の「指定した」とはコマンドライン引数に当該文字列を与えて実行したことを示している．

- ファイル一覧
  存在しないファイル（`absent-file`）を指定した．
- HTTP
  `https://www.google.com` を指定した．
- SCM
  `status` を指定した．いずれの SCM においてもリポジトリではないディレクトリで実行している．
- Clojure インタープリタ
  フィボナッチ数列の 25 項目を求めるスクリプトファイルをそれぞれ指定した（図 1）．

表 3. ファイル一覧の弁別性評価結果

		no-args	help	absent
dir	ls	0.992	0.997	0.992
dir	exa	0.350	0.221	0.215
ls	exa	0.349	0.221	0.215

表 4. HTTP の弁別性評価結果

		no-args	help	google
curl	httpie	0.026	0.032	0.049
curl	wget	0.764	0.342	0.638
httpie	wget	0.030	0.007	0.027

表 5. SCM の弁別性評価結果

		no-args	help	status
Git	Mercurial	0.013	0.013	0.212
Git	Subversion	0.350	0.221	0.215
Mercurial	Subversion	0.349	0.221	0.215

表 6. Clojure インタプリタの弁別性評価結果

		no-args	help	fibonacci
Leiningen	Clojure	0.170	0.789	0.170
Leiningen	Babashka	0.060	0.060	0.060
Leiningen	BBNative	0.058	0.059	0.056
Clojure	Babashka	0.129	0.101	0.130
Clojure	BBNative	0.045	0.097	0.043
Babashka	BBNative	0.211	0.211	0.200

表 3〜6 に実験の結果を示す．各表はカテゴリごとの比較結果を示しており，1, 2 列目は比較したソフトウェア名を示している．BBNative は Babashka のネイティブコードを表している．表 3〜6 の中で高類似度となっているペアは `ls` と `dir` であり，3 項目で共に 0.99 程度になっている．`dir` のソースコードを確認したところ，コメントに `dir` は `ls` のラッパであると記載されており，処理内容は `single_binary_main_ls` を呼び出しているだけであった[4]．

```
(defn fibonacci [n]
  (if (or (= n 0) (= n 1))
    n
    (+ (fibonacci (- n 1)) (fibonacci (- n 2)))))
(println (fibonacci 25))
```

図 1. `fibonacci.clj`（Clojure による fibonacci 数列）

[1] https://github.com/XAMPPRocky/tokei

[2] https://github.com/strace/strace

[3] https://hub.docker.com/r/tamadalab/2023fit_cermeno

[4] https://github.com/coreutils/coreutils/blob/master/src/coreutils-dir.c

つまり，本質的に同じ処理であるため，高い類似度になったと判断できる．

また，`curl` と `wget` が引数なしで実行したときに 0.764，`https://www.google.com` の内容を取得したときに 0.638 と他の値と比べ高い類似度を示している．ただしこの値は，コサイン値であり角度にすると 40.1 度，50.4 度であり，類似性が高いとは言えない．他のペアの類似度を見ても，全て 0.35 以下であり，今回の実験の範囲においては，入力を吟味しなくとも弁別性は確保できると期待できる．

4.3 実験 2 保存性評価

保存性評価では，対象のソフトウェアに対して，実験 1 で用いた 3 入力を与えた．結果を図 2 にヒートマップで示す．図 2 上部に値と色の対応を示している．縦軸，横軸ともに Babashka のバージョンごとに，白の補助線を引いている．そして，各バージョンには入力なし，ヘルプ，フィボナッチ数列の順で並べている．横軸も縦軸と同じ順序で並べている．そのため，対角線は赤になっている．図 2 からわかるように，Babashka 同士の比較は全て高なっている．具体的な数値を見ると，0.4.0〜1.0.164 までの相互比較の類似度は全て 1 であり，1.3.180 とそれ以前のバージョンの類似度は平均して 0.999 であった．一方，Babashka と BBNative の比較結果は全て低くなっており，平均 0.133 であった．また，BBNative 同士の比較ではバージョンが近いもの同士（対角線近く）であれば類似度が高くなっていることがわかる．一方，バージョンが近くなければ類似度が低くなっていることがわかる．

この結果から，GraalVM によるネイティブイメージは，元の Java プログラムとは大きく異なるシステムコールになると伺える．このような方策で盗用が隠された場合は本実験で用いたバースマーク以外の方法が必要となろう．

5. 議論

評価実験では，被告ソフトウェアが拡張された結果，入力の数が変わった場合については考慮していない．例えば，原告では`--help` のようなロングオプションを受け入れていたものを，機能を削り `-h` のようなショートオプションしか受け取らないようにするなどが考えられる．また，指定が任意のオプションを必須のオプションにするなどの変更も考えられる．このような場合，入力のインターフェースが変わるものの，提案手法ではその変化が検知できない．つまり，被告が受け入れない入力を受け取った場合，エラーとなり原告とは異なる実行経路になる．その結果，類似度が低くなると予測できるが，コピー関係にないため類似度が低くなったのか，バースマークへの対策により類似度が低くなったのかを区別できない．より多くの入力を与えることで，一定の対策は可能であると考えられるが，より具体的な対策は今後の課題とする．

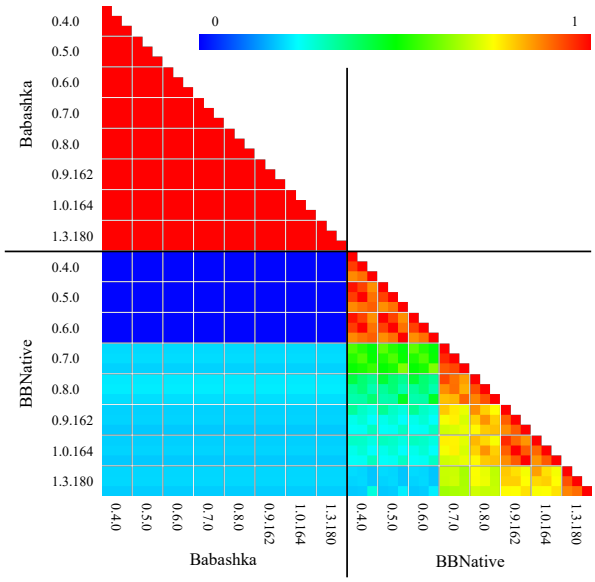

図 2. 保存性評価結果

6. おわりに

本稿では，動的バースマークの自動抽出に向けて，従来の問題になっていた入力を試行錯誤して選択するコストの削減に取り組んだ．解決のために，原告プログラムへの入力を被告プログラムにそのまま与える方法を提案した．評価実験では，同じ機能を持つ複数のソフトウェアに対し同じ入力を与えた．そして，評価実験の範囲においては，バースマークの特性である弁別性と保存性を確認できた．

今後の課題として，より広範囲なソフトウェアへの適用や，他のバースマークを用いての追試，また，攻撃に対する耐性の向上などが挙げられる．

謝辞

本研究の一部は JSPS 科研費 JP20K11761 の助成を受けた．

参考文献

[1] Haruaki Tamada, Masahide Nakamura, Akito Monden, and Kenichi Matsumoto. Java Birthmarks –detecting the software theft–. *IEICE Transactions on Information and Systems*, Vol. E88-D, No. 9, pp. 2148–2158, September 2005.

[2] Shah Nazir, Sara Shahzad, and Neelam Mukhtar. Software birthmark design and estimation: A systematic literature review. *Arabian Journal for Science and Engineering*, Vol. 44, pp. 3905–3927, 2019.

[3] 岡本圭司，玉田春昭，中村匡秀，門田暁人，松本健一．API 呼び出しを用いた動的バースマーク．電子情報通信学会論文誌，Vol. J89-D, No. 8, pp. 1751–1763, August 2006.

[4] Ginger Myles and Christian Collberg. *k*-gram based software birthmarks. Proc. *the 2005 ACM Symposium on Applied Computing* (*SAC 2005*), pp. 314-318, 2005.

[5] Patrick P. F. Chan, Lucas C. K. Hui, and S. M. Yiu. Dynamic software birthmark for Java based on heap memory analysis. In *Communications and Multimedia Security*, pp. 94-107, 2011.

[6] 横井昂典，玉田春昭．単体テストコードとアスペクト指向を用いた動的バースマークの抽出コストの削減．情報処理学会論文誌，Vol. 60, No. 7, pp. 1247–1259, July 2019.

[7] 松田隼汰，神崎雄一郎，光本智洋，玉田春昭．シンボリック実行を利用した動的ソフトウェアバースマークの抽出システムの検討．情報処理学会第 85 回全国大会講演論文集 (講演番号 6K-05), March 2023.

[8] Tian, Z., Zheng, Q., Liu, T. and Fan, M.: DKISB: Dynamic Key Instruction Sequence Birthmark for Software Plagiarism Detection, Proc. *2013 IEEE 10th International Conference on High-Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing* (*HPCC EUC*), pp. 619–627 (2013).

[9] Schuler, D., Dallmeier, V. and Lindig, C.: A Dynamic Birthmark for Java, Proc. *the 22nd IEEE/ACM International Conference on Automated Software Engineering* (*ASE 2007)*, pp. 274–283 (2007).

[10] M. Šipek, B. Mihaljević, and A. Radovan: Exploring Aspects of Polyglot High-Performance Virtual Machine GraalVM, Proc. *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics* (*MIPRO*), May 2019.