# API関数呼び出し履歴を用いた動的ソフトウェアバースマークに対する攻撃への対策

森山 修[†] 古江岳大[†]
遠山 毅[†] 松本 勉[†]

ソフトウェア盗用を発見可能な技術の 1 つとして，ソフトウェアの特徴を利用するソフトウェアバースマークがある．既存研究において岡本らはソフトウェア実行時の API 関数呼び出しの履歴を特徴として用いる手法を提案したが，その手法は API 関数を冗長に呼び出す攻撃に対して脆弱性があることが分かっている．本論文では，バースマークは複数のバースマーク関数を組み合わせて構成されていると考えたうえで，攻撃耐性を考慮したバースマーク関数を提案し，バースマークを構成した．また，提案バースマークの評価実験による有効性確認結果について報告する．

# Counter Measures against the Attacks to Software Dynamic Birthmarks Using History of API Function Calls

Osamu Moriyama,[†] Takahiro Furue,[†] Tsuyoshi Toyama[†] and Tsutomu Matsumoto[†]

Dynamic Birthmarks using runtime information are a class of methods to detect software theft. Okamoto et al. proposed methods that check the history of API function calls. But, Okamoto et al.'s Methods are vulnerable against the attack that calls API functions redundantly. In this paper, after we recognize that Dynamic Birthmark consists of some of functions, we propose new schemes that have resistance against the attack. The effectiveness of our proposal is confirmed by experiments.

## 1. はじめに

ソースコード公開ソフトウェアの不正流用（盗用）に関する問題がたびたび起こっている[10),11)]．また，ソースコードが非公開であっても，リバースエンジニアリングによってソフトウェアが解析され，盗用されるという問題が起こる可能性もある．

そこで，盗用を発見するための技術として，ソフトウェア電子透かし[1)]（以下，電子透かし）や，ソフトウェアバースマーク[2),4),7),18)~20),25)]（以下，バースマーク）が研究されてきた．電子透かしは，ソフトウェアの各モジュールにあらかじめ透かし情報を埋め込んでおき，必要なときに透かし情報を抽出して盗用の有無を検証する技術である．一方バースマークは，2 つのソフトウェア間で，ソフトウェア内の注目する機能を実行するために必要な部分の特徴を比較し，盗用の有無を推測する技術である．

バースマークは電子透かしと異なり，盗用の危険性を考慮せずに配布したソフトウェアについても適用できる．また，電子透かしは透かし情報を消去された場合，盗用の検証ができなくなるが，バースマークは新しい手法が提案されたとき，提案以前に配布したソフトウェアについてもその手法を適用できるため，ある時点で盗用が発見できなくても，後々発見できる場合がある．

ここで，特徴を変化させることにより，盗用を隠蔽する攻撃について考える．既存研究では，ソフトウェア難読化など，既存の等価変換が攻撃の手段として主に考えられてきた．しかし，特徴を変化させることに特化した等価変換なども考えられるため，攻撃への耐性を議論するには，既存の等価変換に着目するだけでは不十分である．また，機能追加など，盗用隠蔽を目的としない改変により結果的に特徴を変化させるという攻撃も考えられる．

本論文では，バースマークの構成を明らかにしたう

† 横浜国立大学大学院環境情報研究院
Graduate School of Environment and Information Sciences, Yokohama National University

えで，API 関数の呼び出しの履歴を用いたバースマークについて述べる．履歴を改変する攻撃への耐性を考慮したバースマークを提案し，その評価実験を行った結果，提案バースマークの有効性を確かめることができた．

## 2. バースマーク

バースマークおよびその周辺の概念の定義を行う．また，既存のバースマークについて述べる．

### 2.1 ソフトウェアの盗用

本論文では，ソフトウェアのライセンスに違反した状態でそのソフトウェアの実装の一部または全体を複製し，他のソフトウェア作成に流用する行為を「盗用」と呼ぶ．また，盗用を行う者を「盗用者」と呼ぶ．

盗用者は，盗用事実の隠蔽を目的にソースコードを改変すると考えられる．また，同時に機能追加・改変を行い，元のソフトウェアとは機能が多少異なったものを作成することも考えられる．ここでは，前者を「盗用隠蔽攻撃」，後者を「機能改変攻撃」と呼び，合わせて「攻撃」と呼ぶ．

### 2.2 バースマーク

「バースマーク」は，ソフトウェア内部の注目する機能を正しく実行するために不可欠である部分の特徴またはその表現を用いて，ソフトウェア間の類似性を測る技術である．なお，この特徴またはその表現を，「バースマーク情報」と呼ぶことにする．ソフトウェア間の類似性は，バースマーク情報の類似量として数値的に測られる．ソフトウェア間の類似性が高いほど，この類似量は大きくなる．すなわち，バースマークを用いて盗用を発見するということは，ソフトウェア間の類似性を類似量として測ることにより，ソースコード流用の有無を推測するということである．

バースマークは，2 つのソフトウェア $P$，$Q$ について，次の 2 性質を満たすことが望ましい．ただし $B_P$，$B_Q$ は $P$，$Q$ のバースマーク情報とする．

- **攻撃耐性** $P$，$Q$ の実装の類似性が高いとき，$B_P$，$B_Q$ の類似量は有意に大きい．
- **弁別性** $P$，$Q$ の，ユーザから見た振舞いの類似性が高くても，実装の類似性が低いとき，$B_P$，$B_Q$ の類似量は有意に小さい．

攻撃耐性は，既存研究[20] において「保存性」と呼ばれているものに相当するが，本論文ではバースマーク情報を変化させる攻撃への耐性を特に考慮するため，このように呼ぶことにする．また，類似量がどの程度であれば流用されていると判断できるかについては，様々なソフトウェアから類似量を求めた結果から統計的に与えられるものとし，本論文では閾値などを設定せず，傾向に着目する．

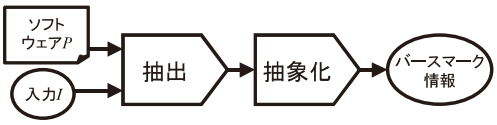


**図 1** 古田らのバースマーク情報構成図
Fig. 1 Furuta et al.'s construction of birthmarks info.

バースマークは次の 2 種類に大きく分類される．

- **静的バースマーク** ソフトウェアを実行せずに得られる静的情報を利用するバースマーク．
- **動的バースマーク** ソフトウェアの実行過程に現れる動的情報を利用するバースマーク．

また，これらのバースマーク情報を，それぞれ静的バースマーク情報，動的バースマーク情報と呼ぶ．

### 2.3 バースマークの構成

バースマークには盗用証拠能力が低いなどの短所が存在するため，複数のバースマークを用いて短所を補うべきである．しかし，これまでに提案されているバースマークの種類は少なく，短所を補うには不十分である．その問題に対し古田ら[2] は，動的バースマーク情報の生成手順は抽出と抽象化の 2 つから構成されており，それらを様々に組み合わせることによって新たなバースマーク情報を作り出せる，と主張している．抽出とは，ソフトウェア $P$ とその入力 $I$ に基づく実行過程の全情報からある特徴を持つ要素を抜き出し，実行順に並べた系列を得る操作を指す．この系列を抽出系列といい，これが $P$ の特徴にあたる．また，抽象化とは，抽出系列をもとに何らかの抽象化表現を得る操作を指す．この抽象化表現が攻撃耐性と弁別性を持つならばそれがバースマーク情報となる．これを図に示したものが，**図 1** となる．

古田らは 1 つのソフトウェアから動的バースマーク情報を得るまでの処理のみに着目している．そこで，これを拡張し，バースマークの構成を新たに提案する．

拡張の 1 つ目として，抽出処理を静的バースマークに対しても考え，抽出される情報を「抽出情報」と呼び，抽出情報を構成する成分を「抽出成分」と呼ぶことにする．抽出対象ソフトウェアに対する入力が $\phi$ である場合は静的バースマークに相当し，$\phi$ でない場合は動的バースマークに相当する．ここで，$\phi$ は何もないことを示す記号とする．また，動的バースマークにおいて，抽出情報が実行順序に関する情報を含んでいる場合，抽出系列と見なすことができる．

2 つ目として，バースマーク情報間の類似量導出方法に着目する．類似量導出方法の選択により，類似量

には様々な傾向が現れると考えられる．そこで，類似量導出もバースマーク構成要素として考える．

3つ目として，抽出と抽象化の間に着目する．これまでに，抽出，抽象化，類似量導出という3つの処理について述べた．抽出から抽象化までが，1つのソフトウェアから1つのバースマーク情報を得るまでの処理であり，この間，2つのソフトウェアに由来するデータどうしが絡むことはない．しかし，たとえば片方の抽出情報に含まれる種類の抽出成分以外は削除するなど，抽象化の前処理として2つのソフトウェアから得られた各抽出情報を比較し操作するという処理を行うバースマークも考えられる．類似量導出処理においてこのような処理を担当しようとしても，入力は抽象化表現であるため抽出情報を直接操作することはできない．また，この比較・操作処理も他の3処理と同様に，どのような処理をするか選択することにより，バースマーク情報や類似量に様々な傾向が現れると考えられる．そこで，抽出情報の比較・操作処理もバースマーク構成要素として別個に考える．これにより，提案バースマーク構成の一般性を向上させることができる．

以上の拡張から，本論文の筆者らは，バースマークは以下に示す4種類の関数から構成されていると考える．ただし，$P$，$Q$ は比較する2つのソフトウェアであり，$I$ は $P$，$J$ は $Q$ の，比較すべき実行状態を導くための入力である．

- **抽出関数 Ext**　$P$ と $I$，または $Q$ と $J$ を入力とし，特徴である抽出情報を出力する．$I$，$J$ が $\phi$ であるときは静的バースマークに相当し，$I$，$J$ が $\phi$ でないときは動的バースマークに相当する．$I$，$J$ の片方だけが $\phi$ であることは許さない．
- **抽出情報比較関数 Cmp**　2つの抽出情報を入力とし，2つの処理済み抽出情報を出力する．
- **抽象化関数 Abs**　1つの抽出情報を入力とし，1つの抽象化表現を出力する．
- **類似量導出関数 Sim**　2つの抽象化表現を入力とし，それらの類似量を出力する．

この4種類の関数それぞれを「バースマーク関数」と呼ぶ．また，各バースマーク関数の入出力を「バースマーク変数」と呼ぶ．なお，各バースマーク関数には，関数の振舞いを制御するための入力も存在する場合があると考えられる．このような入力を「制御入力」と呼ぶ．

以上のバースマーク構成を図にしたものが**図2**である．ただし，制御入力については図の簡潔化のため，省略している．なお，図2ではExt，Absが2つずつ表記されているが，それぞれ同じExt，Absを用いるものとする．

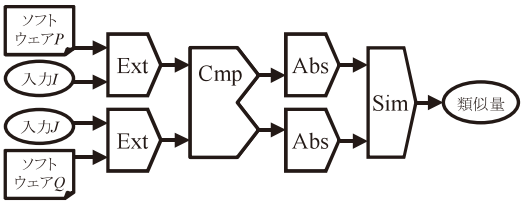


**図2** 提案するバースマーク構成図
Fig. 2 Proposing construction of birthmarks.

本論文では，これ以降動的バースマークに特化して議論を進める．すなわち，ソフトウェアへの入力が $\phi$ であることを許さないExtを用いるものとする．また，抽出情報を抽出系列として扱うものとする．

### 2.4 既存の動的バースマーク

すでに，動的バースマークとしていくつかの手法が提案されている．岡本ら[20]はAPI関数の呼び出し順序・頻度を用いた実行系列バースマーク（以下DB_SEQ），実行頻度バースマーク（以下DB_FREQ）を，Mylesら[18]はWhole Program Paths[12]を用いたバースマーク（以下DB_WPP）を，それぞれ独立に提案している．また，古田ら[2]はある特徴の繰返しパターンを抽象化表現とする動的バースマーク（以下DB_REP）を，林ら[7]はある特徴の出現位置に着目して抽象化を行う出現間隔バースマーク（以下DB_INTV），近傍検索バースマーク（以下DB_SRCH）を提案している．なお，これらのバースマーク関数を**表1**にそれぞれ記述した．

岡本ら[20]が提案した，API関数の呼び出し履歴を抽出系列として用いる動的バースマークは，API関数の実体がソフトウェアの外部に存在する場合が多いため，ソフトウェアに難読化などが施されていてもその実体は変わりえず，また呼び出しの方法も決まっており，呼び出しの隠蔽は難しいと考えられるということを攻撃耐性の根拠にしている．本論文ではその考えに従い，共有ライブラリにその実装を持つAPI関数呼び出し履歴を抽出系列として選択し，議論を進める．

なお，DB_WPPは，API関数呼び出しを抽出成分とすることは考慮されておらず，DB_SRCHは抽出成分がAPI関数呼び出しのみでは適用が難しい．これらはAPI関数呼び出し履歴を抽出系列とする前提にはそぐわないため，本論文ではこの2つについて議論しない．

## 3. 動的バースマークに対する攻撃

ここでは盗用者の能力を仮定し，盗用者が行う攻撃についてまとめる．

**表 1** 既存の動的バースマーク
Table 1 Existing dynamic birthmarks.

| 名称 | Ext | Cmp | Abs | Sim |
|---|---|---|---|---|
| DB_SEQ | E_api | C_nop | A_nop | S_lms |
| DB_FREQ | E_api | C_nop | A_vec | S_vec |
| DB_WPP | E_wpp | C_nop | A_wpp | S_mcs |
| DB_REP | E_ucid | C_nop | A_rep | S_lms |
| DB_INTV | E_ucid | C_nop | A_intv | S_lms |
| DB_SRCH | E_id | C_nop | A_srch | S_and |

E_api : API 関数の実行順系列を出力
E_wpp : 処理ブロックの実行順系列を出力
E_ucid : 難読化などで変化しない識別子（不変な名前）の実行順系列を出力
E_id : ある種類の識別子の実行順系列を出力
C_nop : 入力をそのまま出力
A_nop : 入力をそのまま出力
A_vec : 入力中に含まれる成分を種類ごとに数え，その数を成分とする多次元ベクトルを出力
A_wpp : Whole Program Paths と呼ばれるグラフを出力
A_rep : 入力の繰返しパターンを用いて抽象化し，系列を出力
A_intv : ある 1 種の識別子の，系列中での出現間隔を用いて抽象化し，系列を出力
A_srch : ある識別子から距離 d 内にある不変な名前より集合を生成
S_lms : 2 系列の最長一致列長から類似量を計算
S_vec : 2 ベクトルの成す角から類似量を計算
S_mcs : 2 グラフの最大共通部分グラフから類似量を計算
S_and : 2 集合の積集合から類似量を計算

### 3.1 盗 用 者

盗用者の目的は，盗用事実を発見されないように盗用を行うことである．盗用者の能力および行動を，以下のように仮定する．

(ア) 盗用対象ソフトウェア $P$ のソースコードの記述をすべて参照できる．
(イ) 盗用隠蔽を目的とする行動にかけるコストは，$P$ と同機能のソフトウェアを自作するためにかかるコストよりも有意に低い．
(ウ) $P$ のバースマーク情報 $B_P$ と，$P$ を盗用して作成したソフトウェア $Q$ のバースマーク情報 $B_Q$ をそれぞれ求めることができ，類似量を導出することができる．
(エ) $Q$ は公開し，$Q$ のソースコードは公開しない．また，$Q$ は逆コンパイルが困難である．

(ア) は，ソースコードが公開されているか，または $P$ の逆コンパイルが可能であることを示す．(イ) は，盗用隠蔽のためにはコストをかける必要があり，またかけられるコストは限られていることを示す．これは，$P$ と同機能のソフトウェアを自作可能ならば，盗用する必要がないという考えに由来する．(ウ) は，$Q$ の作成時に存在するバースマークを用いて $P$ と $Q$ のバースマーク情報の類似量を求め，攻撃の指標にできることを示す．攻撃と類似量導出を繰り返す回数は (イ) によって制限される．(エ) は，盗用発見のために $Q$ のソースコードは利用不可能であることを示す．

**表 2** 盗用隠蔽攻撃の分類
Table 2 Classification of theft concealment attack.

| 基本操作 | 攻撃名 | 詳細分類 |
|---|---|---|
| 削除 | API 関数削除攻撃 | 自作攻撃 |
| | | 複製攻撃 |
| 挿入 | API 関数挿入攻撃 | |
| 置換(+ 削除・挿入) | API 関数置換攻撃 | |
| (2 回以上) | API 関数順序入替え攻撃 | |

### 3.2 攻撃の最小単位

盗用者にとって攻撃とはソースコードを改変することである．攻撃により最も影響を受けやすいバースマーク変数は抽出情報である．

抽出情報は抽出系列であるとする．抽出系列を文字列としてとらえることにすると，抽出系列を変化させる攻撃の最小単位は，文字列の基本操作である，1 文字の削除・挿入・置換の 3 種類と考えることができる．以降，1 回の攻撃は，抽出系列に対する 1 回の基本操作に相当するものとする．

### 3.3 盗用隠蔽攻撃

盗用隠蔽攻撃は，盗用の隠蔽を目的とする攻撃である．本論文では，盗用隠蔽攻撃はソフトウェアが持つ機能を変更せずに，API 関数呼び出しを変更する攻撃を指すものとする．

API 関数呼び出し履歴を抽出系列と見なしたとき，それに対する攻撃を，攻撃の最小単位に基づいて考えることができる．それをまとめたものが**表 2** である．各攻撃について説明する．

API 関数削除攻撃とは，API 関数の機能をユーザ関数として実装することにより API 関数呼び出しを消す攻撃である．方法として，自作攻撃と複製攻撃があげられる．自作攻撃は，API 関数が持つ機能を盗用者が自ら実装する攻撃であり，複製攻撃は，API 関数の実装を丸ごと複製し利用する攻撃である．

API 関数挿入攻撃は，API 関数を冗長に呼び出し，履歴を変化させる攻撃である．

API 関数置換攻撃は，ある機能を実現している API 関数の組を，同等機能が実現可能な別の API 関数の組に置換し，履歴を変化させる攻撃である．この攻撃

**表 3** 盗用隠蔽攻撃のコスト
Table 3 Cost of theft concealment attack.

| 攻撃略称 | | 準備時コスト | 改変時コスト | 攻撃可能回数 |
|---|---|---|---|---|
| 削除 | 自作 | 対象 API 関数の処理を解析，同等処理を実装するコスト | 特に無し | 大 |
| | 複製 | 対象 API 関数の実装を解析，複製するコスト他 | 特に無し | 大 |
| 挿入 | | API 関数に関する知識の習得 | 特に無し | 大 |
| 置換 | | API 関数に関する知識の習得 | 置換可能箇所の探索, 置換方法の考案 | 小 |
| 順序入替 | | API 関数に関する知識の習得 | 順序入替え可能箇所の探索 | 小 |

**表 4** 盗用隠蔽攻撃への耐性
Table 4 Resistance to theft concealment attack.

| バースマーク | 盗用隠蔽攻撃耐性 | | | | |
|---|---|---|---|---|---|
| | 耐自 | 耐複 | 耐挿 | 耐置 | 耐順 |
| DB_SEQ | *1 | *2 | × | × | × |
| DB_FREQ | *1 | *2 | × | △ | ○ |
| DB_REP | *1 | *2 | × | × | × |
| DB_INTV | *1 | *2 | × | × | × |

耐自：API 関数自作攻撃耐性　耐複：API 関数複製攻撃耐性
耐挿：API 関数挿入攻撃耐性　耐置：API 関数置換攻撃耐性
耐順：API 関数順序入替え攻撃耐性
○：攻撃影響無　△：攻撃影響小　×：攻撃影響大
*1：自作が難しい API 関数のみを抽出対象とすれば対策可能[20]であるが，未評価.
*2：API 関数の実装複製が実際に可能であれば，すべて“×”となる.

は置換操作だけでなく，挿入・削除操作を組み合わせる場合が多いと考えられる．

API 関数順序入替え攻撃は，2 回以上の基本操作を組み合わせたもので，呼び出し順序が処理に影響のない箇所を探し，順序を入れ替える攻撃である．

これらの攻撃を行うためには，3.1 節，盗用者の能力 (イ) で述べたとおり，コストがかかる．このコストをまとめたものが**表 3** である．ここで攻撃のコストとは，盗用者が対象ソースコードおよび API 関数について意味解析を行うときにかかるコストとする．準備時コストとは盗用者が事前に行う必要のある行動にかかるコストであり，改変時コストとは対象ソースコードを改変する際にかかるコストである．また，API 関数削除攻撃は，ある API 関数をユーザ関数として実装することに成功した場合，その API 関数呼び出しはすべて削除することができるようになるため，攻撃可能回数は大きいと考えられる．API 関数挿入攻撃についても，処理に影響を与えないように呼び出しを行えば，何回でも呼び出しができるため，攻撃可能回数は大きいと考えられる．残り 2 つの攻撃は攻撃対象ソフトウェアの実装に依存するところが大きいが，攻撃可能回数は比較的小さいと考えられる．

### 3.4 機能改変攻撃

機能改変攻撃は盗用の隠蔽を目的とせず，機能の追加，改変などの結果，API 関数呼び出し履歴が変化する攻撃である．そのため，この攻撃によってソフトウェアの機能には変更が加えられる．例としては，余分な機能の削除や部分的な盗用，新機能の追加，機能の置換などがあげられる．

機能改変攻撃に関しては，攻撃能力の制限を設けることが難しいため，本論文では特に制限を設けない．すなわち，機能改変攻撃の程度は盗用者の自由となる．

### 3.5 既存バースマークの攻撃耐性

盗用隠蔽攻撃と機能改変攻撃に対し，既存のバースマークがどの程度の攻撃耐性を持っているか，攻撃コストを考慮した評価を行った．

#### 3.5.1 盗用隠蔽攻撃への耐性

盗用隠蔽攻撃への耐性についてまとめたものが**表 4** である．以下，各評価理由について述べる．

API 関数複製攻撃および API 関数挿入攻撃については，攻撃可能回数が多く，改変時コストがかからないため，多数回攻撃されても影響を小さく抑えられるかどうかが評価基準となる．API 関数複製攻撃については，この攻撃が可能であればすべての API 関数の抽出ができず，バースマーク情報の導出ができなくなる．API 関数挿入攻撃については，既存の動的バースマークにおいて対策が施されているものは存在しない．

API 関数置換攻撃および API 関数順序入替え攻撃については，攻撃時コストがかかるため，少ない攻撃回数でも大きな影響が与えられる可能性があるかどうかが評価基準となる．DB_SEQ，DB_REP，DB_INTV の類似度導出関数 S_lms で用いられている最長一致列長は，2 つの系列のうち，最も長く一致している部分の長さである．したがって，最長一致列の中心の 1 カ所が変化しただけで，その長さは半分になる．DB_SEQ は抽象化を行っていないため，抽出系列の操作が直接類似量に現れる．よって，API 関数置換攻撃，API 関数順序入替え攻撃による影響は大きいと考えられる．DB_REP および DB_INTV はそれぞれ抽象化を行っているが，それでも大きく影響を受けると考えられる．

**表 5** 機能改変攻撃への耐性
Table 5 Resistance to function modification attack.

| バースマーク | 機能改変攻撃耐性 |
|---|---|
| DB_SEQ | × |
| DB_FREQ | ○ |
| DB_REP | × |
| DB_INTV | × |

○：類似性測定良好 ×：類似性測定不良

DB_FREQ については，置換攻撃の影響は受けるものの，少ない回数の置換攻撃では API 関数の呼び出し総数への影響は小さいと考えられる．また，系列順序はバースマーク情報の導出に用いないため，実行頻度は順序入替え攻撃の影響を受けない．

#### 3.5.2 機能改変攻撃への耐性

バースマークはソフトウェア間の類似性を類似量として測る技術であるため，機能改変によって API 関数呼び出し履歴が変化した場合，その変化量に応じて類似量も変化すべきである．すなわち，履歴の変化がわずかであるにもかかわらず類似量が非常に小さくなったり，逆に履歴のほとんどが変化しているにもかかわらず類似量が大きいままであったりすることは好ましくない．さらにいえば，履歴の変化量と類似量は比例関係に近い方が好ましい．以上から，機能改変攻撃への耐性とは，ソフトウェア間の類似性を正しく測れるかどうかに相当する．このような基準に基づいて評価を行った結果をまとめたものが**表 5** である．

DB_SEQ は S_lms を用いている．3.5.1 項で述べたように，最長一致列長は系列の改変に弱い．よって，呼び出し履歴の類似性は高くても，類似量が小さくなってしまう場合が考えられるため，ソフトウェア間の類似性が正しく測れない場合があると考えられる．

DB_REP，DB_INTV も S_lms を用いている．この 2 つのバースマークは抽象化を行っているが，それでも抽出系列の改変による類似量への影響は大きく，ソフトウェア間の類似性が正しく測れない場合があると考えられる．

DB_FREQ では S_vec を用いている．履歴の変化量と類似量は比例関係にあるとはいえないが，S_lms のような急峻な変化はせず，ソフトウェア間の類似性は比較的正しく測れると考えられる．

## 4. 提　　案

既存手法と比べ，攻撃への耐性を向上させることを目的に，新たなバースマーク関数およびバースマークを提案する．表 4 より，API 関数挿入攻撃への耐性を持つ動的バースマークは未提案であると分かる．また，3.5.2 項より，S_lms は機能改変攻撃への耐性を持たないことが分かる．そこで，本章では API 関数挿入攻撃への耐性や機能改変攻撃への耐性を考慮したバースマークおよびバースマーク関数を提案する．

### 4.1 失敗呼び出し削除を行う抽出関数の提案

API 関数挿入攻撃においてどのような API 関数が挿入されるかを考える．API 関数挿入攻撃は盗用隠蔽攻撃であるため，ソフトウェアの挙動に影響を与えないように API 関数が挿入される．いい換えれば，処理に影響を与えない API 関数呼び出し（以下，無効呼び出し）が挿入されると考えられる．よって，無効呼び出しを抽出時に検知し，排除すればよい．

盗用者が無効呼び出しを行う方法としては，API 関数の実行に失敗するような引数を渡して呼び出すという方法（以下，失敗呼び出し）や，機能 F とその逆の機能 $F^{-1}$ を持つ 2 種類の API 関数を連続的に呼び出し，機能を打ち消すという方法（以下，打消呼び出し），そもそも処理に影響を与えない API 関数を呼び出す方法（以下，正常無効呼び出し）などが考えられる．

どのような無効呼び出しを行うかについては盗用者が決定するため，攻撃対策としてはすべての無効呼び出しに対応すべきである．しかし，前述した 3 種の無効呼び出し以外にも未判明の無効呼び出しの存在が考えられるため，すべての無効呼び出しに対応することは難しい．本論文では，無効呼び出し削除による挿入攻撃対策の入口として，API 関数の種類に依存する部分の少ない無効呼び出し検知方法を考える．

32 bit Windows における API 関数の多くは，実行失敗時に SetLastError 関数などを用いてエラーコードを設定する．したがって，API 関数の呼び出しごとにエラーコードを監視する，という API 関数の種類に依存する部分が少ない方法で失敗呼び出し検知が可能になる．そして，検知された失敗呼び出しを履歴として採用しなければ，攻撃者による API 関数挿入攻撃の手段を減らすことができ，耐性が向上する．この抽出関数を E_fdel と呼び，API 関数挿入攻撃に耐性を持つ抽出関数として提案する．

なお，この処理は Abs 内の処理であると考えることもできるが，本論文では Ext 内の処理として扱う．

### 4.2 編集距離を用いた類似量導出関数の提案

バースマーク情報が系列である既存動的バースマークの類似量導出関数 S_lms は，以下の式で表される．

$$\mathrm{S\_lms}(S_1, S_2) = \frac{2 \times \mathrm{LMS}(S_1, S_2)}{\mathrm{Len}(S_1) + \mathrm{Len}(S_2)} \qquad (1)$$

ただし，$S_1$，$S_2$ は比較する 2 つの系列であり，LMS

は 2 つの系列の最長一致列長を，Len は系列の長さを返す関数である．

S_lms では，全体に細かく改変が施された場合，元々は最長一致列であった箇所が細切れになり，類似量が著しく小さくなってしまう．そこで本論文では，編集距離[3] を用いた系列類似量の導出法を提案する．編集距離は，ある文字列を別の文字列に変えるとき，文字列の基本操作である，1 文字の削除・挿入・置換を最小で何回行えばよいかを示す値で，これを用いて 2 つの文字列の差を定量的に測ることができる．編集距離で用いられる基本操作は，3.2 節で述べた攻撃の最小単位と等しい．したがって，少ない回数の攻撃では編集距離は大きく影響を受けない．そこで，この編集距離を用いて系列類似量を導出する．

編集距離を用いた類似量導出関数 S_ed を，以下のように提案する．

$$\mathrm{S\_ed}(S_1, S_2) = 1 - \frac{\mathrm{EditDist}(S_1, S_2)}{\mathrm{MaxLen}(S_1, S_2)} \qquad (2)$$

ただし，$S_1$，$S_2$ は比較する 2 つの系列であり，EditDist は 2 つの系列の編集距離を，MaxLen は 2 つの系列のうち長い方の系列長を返す関数である．

編集距離は 1 文字の削除・挿入・置換を行う回数を示している．そのため，API 関数挿入攻撃を行った回数だけ編集距離は増加する．そこで，編集距離のうち，挿入に由来する距離を打消すことができれば，API 関数挿入攻撃に耐性を持つ系列類似量導出関数を実現することができると考えられる．そこで，比較する 2 系列の長さの差を打ち消すように S_ed を改めた関数 S_ed2 を以下のように提案する．

$$\mathrm{S\_ed2}(S_1, S_2) = 1 - \frac{\mathrm{EditDist}(S_1, S_2) - \mathrm{LenDiff}(S_1, S_2)}{\mathrm{MinLen}(S_1, S_2)} \qquad (3)$$

ただし，$S_1$，$S_2$ は比較する 2 つの系列であり，EditDist は 2 つの系列の編集距離を，LenDiff は 2 つの系列長の差分を，MinLen は 2 つの系列のうち短い方の系列長を返す関数である．この式は，S_ed の右辺第 2 項の分子・分母からそれぞれ LenDiff を引いたものに相当する．

以上より，機能改変攻撃に耐性を持つ類似量導出関数として S_ed および S_ed2 を，API 関数挿入攻撃に耐性を持つ類似量導出関数として S_ed2 を提案する．

ここで，式 (2)，式 (3) をそれぞれ変形すると，以下のようになる．

$$\mathrm{S\_ed}(S_1, S_2) = \frac{\mathrm{MaxLen}(S_1, S_2) - \mathrm{EditDist}(S_1, S_2)}{\mathrm{MaxLen}(S_1, S_2)} \qquad (4)$$

$$\mathrm{S\_ed2}(S_1, S_2) = \frac{\mathrm{MaxLen}(S_1, S_2) - \mathrm{EditDist}(S_1, S_2)}{\mathrm{MaxLen}(S_1, S_2) - \mathrm{LenDiff}(S_1, S_2)} \qquad (5)$$

式 (4)，式 (5) を比較すると，これらは分子が等しく，分母が LenDiff だけ異なっている．したがって，両式の $S_1$，$S_2$ が共通の場合，S_ed2 で求まる類似量は，S_ed で求まる類似量に比べ，大きくなるといえる．よって，S_ed2 は S_ed よりも弁別性が低いと考えられる．

### 4.3 *k*-gram を利用した動的バースマークの提案

Myles らによって，*k*-gram を用いた静的バースマーク（k-gram Based Software Birthmarks，以下 kBSB）が提案されている[19]．以下，kBSB について述べる．

*k*-gram とは，系列中に隣接して現れる $k$ 個の記号の組である．たとえば “ABCDC” という系列から 3-gram の組すべてを得ると，(ABC)，(BCD)，(CDC) となる．

ソフトウェア内のオペコードの並びは系列として扱えるため，そこよりすべての *k*-gram の組を求める．そして，重複する *k*-gram を削除し，*k*-gram 集合を得る．この *k*-gram 集合をバースマーク情報とし，この抽象化関数を A_kg と呼ぶ．2 つのソフトウェアから得た *k*-gram 集合より，以下の類似量導出関数 S_kg を用いて類似量を求める．

$$\mathrm{S\_kg}(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1|} \qquad (6)$$

ただし，$S_1$ と $S_2$ は比較する 2 つの *k*-gram 集合であり，$|S_1|$，$|S_2|$ は $S_1$，$S_2$ の元の数を表す．なお，$S_1$ は比較する元となるソフトウェアから，$S_2$ は比較対象となるソフトウェアから抽出された *k*-gram 集合であるとする．すなわち，あるソフトウェア A と，A を盗用して作られた疑いのあるソフトウェア B を比較するとき，A から得られた *k*-gram 集合を $S_1$ に，B から得られた *k*-gram 集合を $S_2$ にあてはめ，S_kg を用いることになる．

kBSB は静的バースマークであり，API 関数呼び出し履歴を意識して提案されたものではないが，kBSB におけるオペコードを API 関数呼び出しと見なせば，動的バースマークとして利用することができる．これを DB_KBSB と呼ぶ．

ここで，盗用者が改変することのできる *k*-gram 集合は $S_2$ だけであり，$S_1$ は改変できないことに着目する．S_kg の分母は $S_1$ の元の数，分子は $S_1 \cap S_2$ の元の数であるため，盗用者が S_kg の出力値を小さくするために可能なことは，分子の値を小さくすること，

すなわち，$S_2$ から $S_1$ に含まれる $k$-gram を取り除くことのみである．API 関数挿入攻撃でこれを達成するためには，たとえば，3-gram 集合から取り除きたい 3-gram を (ABC) とすると，API 関数呼び出し履歴に含まれるすべての (ABC) を (AXBC) などのようにする必要がある．これは盗用者にかかる改変コストが既存の動的バースマークよりも高くなることを意味している．したがって，DB_KBSB は API 関数挿入攻撃に対する耐性が既存動的バースマークよりも向上していると考えられる．そこで，API 関数挿入攻撃に耐性を持つバースマークとして，DB_KBSB を提案する．

## 5. 評価実験および考察

これまでに述べた提案をもとに評価実験を行い，結果の考察を行った．

### 5.1 実験の目的

4 章で述べた各提案関数の評価を目的とし，実験を行う．

4.1 節では失敗呼び出しは無効呼び出しであるとした．しかし，API 関数の実行失敗によって処理を分岐するソフトウェアも存在する．すなわち，実行に失敗した API 関数呼び出しが必ず無効呼び出しであるとはいえない．よって，失敗呼び出しを履歴から削除した場合，有効な API 関数呼び出しまで削除するため，機能改変攻撃耐性や弁別性に影響が現れると考えられる．そこで，類似量にどの程度変化が現れるか確認する．

次に，S_ed および S_ed2 は S_lms と比べて機能改変攻撃耐性がどの程度向上しているか，また弁別性はどの程度存在するかを確認する．さらに，S_ed2 の API 関数挿入攻撃耐性についても確認する．

最後に，DB_KBSB は実際にどの程度 API 関数挿入攻撃に対して耐性があるか，また機能改変攻撃耐性や弁別性は存在するか確認する．

### 5.2 実験対象ソフトウェア

実験対象のソフトウェアとして，**表 6**，**表 7** に示す 2 種類のソフトウェア群を用いる．表 6 は 2 ちゃんねるブラウザと呼ばれるソフトウェア群である．このソフトウェア群を用いた理由として，ソフトウェア A のソースコードが公開されており，その派生物（$B_1$，$B_2$，以下 $B_i$）が複数存在することや，A と主目的が同じだが A の派生物ではないソフトウェア（$C_1$，$C_2$，$C_3$，以下 $C_i$）が複数存在することなどがあげられる．A と $B_i$ とは類似量が大きく，$C_i$ とは小さくなることが望ましい．また，A' は A に対して API 関数挿入攻撃を施したものである．本実験では，エラーコードを設定する 5 種類の API 関数をランダムに呼び出す関数を作成し，A のソースコードの各ステップに挿入し，挿入攻撃を達成した．

**表 6** 実験対象ソフトウェア群 1
Table 6 Group of software for Experiment 1.

| 表記 | ソフトウェア名 | バージョン | 入手先 |
|---|---|---|---|
| A | Open Jane Doe | ※ | 21) |
| A' | Open Jane Doe (挿入攻撃) | ※ | 21) |
| $B_1$ | Jane Doe View | 060806a | 26) |
| $B_2$ | Jane Doe Style | 2.41 | 24) |
| $C_1$ | A Bone | 1.50 | 9) |
| $C_2$ | えまのん | R20060608 | 22) |
| $C_3$ | ギコナビ | バタ 53 | 17) |

※ 2006 年 8 月 12 日にソースコードリポジトリ [21] から入手したソースコードを，Borland Delphi6 Personal Update Pack2 を用いてコンパイルし，利用した．

**表 7** 実験対象ソフトウェア群 2
Table 7 Group of software for Experiment 2.

| 表記 | ソフトウェア名 | バージョン | 入手先 |
|---|---|---|---|
| $\alpha$ | Super Tag Editor | 2.02 | 15) |
| $\beta_1$ | Super Tag Editor 改 | Hurricane3 | 5) |
| $\beta_2$ | Super Tag Editor 改 Plugin Version | 1.02 | 6) |
| $\beta_3$ | kbSTE | 2.02d | 13) |
| $\gamma_1$ | Mp3tag | 2.36a | 8) |
| $\gamma_2$ | Teatime | 2.525 | 14) |
| $\gamma_3$ | つゆたぐ | 2.04 | 23) |

表 7 は MP3 タグエディタと呼ばれるソフトウェア群である．$\alpha$ はソースコード公開のソフトウェアであり，$\beta_1$，$\beta_2$，$\beta_3$（以下 $\beta_i$）は $\alpha$ の派生物，$\gamma_1$，$\gamma_2$，$\gamma_3$（以下 $\gamma_i$）は $\alpha$ と主目的が同じであるが派生物ではないソフトウェアである．$\alpha$ と $\beta_i$ とは類似量が大きく，$\gamma_i$ とは小さくなることが望ましい．$\beta_2$，$\beta_3$ 以外については，岡本ら [20] がバースマークについての実験に用いたソフトウェアであり，本実験ではこれらについても実験を行った．

なお，A，$\alpha$ は各派生物の派生元の版ではなく，実験時に公開されていた最新版を用いているが，この場合においてもそれぞれの類似性は高いと期待できる．

### 5.3 履歴取得対象 API 関数

呼び出し履歴取得対象の API 関数として，Microsoft Windows Server 2003 SP1 Platform SDK Version 3790.1830 [16] に含まれる WinBase.h ヘッダファイル

**表 8** 提案バースマーク関数の組合せ
Table 8 Combination of proposed birthmarks function.

| 表記 | Ext | Abs | Sim |
|---|---|---|---|
| DB_SEQ | E_api | A_nop | S_lms |
| DB_SEQ_FDEL | E_fdel | A_nop | S_lms |
| DB_SEQ_ED | E_api | A_nop | S_ed |
| DB_SEQ_FDEL_ED | E_fdel | A_nop | S_ed |
| DB_SEQ_ED2 | E_api | A_nop | S_ed2 |
| DB_SEQ_FDEL_ED2 | E_fdel | A_nop | S_ed2 |
| DB_KBSB | E_api | A_kg | S_kg |
| DB_KBSB_FDEL | E_fdel | A_kg | S_kg |

※ ただし，Cmp はすべて C_nop

において WINBASEAPI マクロを用いてプロトタイプ宣言されている API 関数 656 種のうち，実験環境では用いることのできない 29 種と，実験対象ソフトウェアすべてに共通して呼び出し回数が膨大である 4 種を除いた 623 種を用いた．呼び出し回数が膨大である 4 種の API 関数については，その呼び出しの影響で，履歴の記録に支障が出る場合があることと，膨大な回数の呼び出しが 1 カ所にまとまって現れる傾向にあるため，バースマーク情報が系列であるバースマークの弁別性が著しく下がる可能性があるという 2 つの理由により，本実験では履歴取得対象として含めない．

### 5.4 実験手順

実験対象ソフトウェアを起動させ，起動と同時に API 関数呼び出し履歴の取得を開始する．起動時の処理が終了した後，すぐにウィンドウを閉じ，終了させる．終了処理が終了した後，履歴取得を終了する．これをすべてのソフトウェアに対して行う．

呼び出し履歴は，API 関数名と呼び出し元スレッド ID，エラーコードからなっており，これが実行順に並んでいる．得られた呼び出し履歴はスレッドごとの呼び出しが混在しているため，スレッド ID を用いて呼び出し元スレッドごとにソートする．

ソート済みの履歴データを用い，各提案バースマーク関数によって類似量を求める．ここで，バースマーク関数は単体では効力を持たず，Ext，Cmp，Abs，Sim の 4 つが揃って初めて類似量を計算できる．そこで，E_fdel，E_ed，E_ed2 に関しては，**表 8** のようにバースマーク関数を組み合わせ，実験を行う．DB_SEQ は提案バースマークではないが，比較対象とするため実験を行う．なお，エラーコードが履歴に記録されているため，履歴取得後に失敗呼び出しを削除することもできる．そこで本実験では E_api の出力から失敗呼び出しを削除した系列を，E_fdel の出力とする．また，内部で別の API 関数を呼び出す API 関数が失敗したときは，それらすべての呼び出しを失敗呼び出しと見なす．

DB_KBSB は，制御入力によって $k$ の値を選択できる．$k$ の値によって $k$-gram 集合が変化するため，$k$ を 1 から 9 まで変化させたときの類似量をそれぞれ求める．なお，DB_KBSB においても，抽出関数として E_api，E_fdel を選択することができる．そこで，この両方について実験を行う．抽出関数に E_fdel を選択したものを，DB_KBSB_FDEL と呼ぶ．

実験環境は，マシンに Gateway GT4014J（Athlon64 X2 3800+，1 GB RAM），OS に Microsoft Windows XP Home Edition SP2 を用いる．

### 5.5 実験結果

E_fdel，S_ed，S_ed2 についての実験結果を，**表 9**，**表 10** に示した．表 9 はソフトウェア群 1 に対する実験結果で，A とその他のバースマーク情報の類似量を示す．表 10 はソフトウェア群 2 に対する実験結果で，$\alpha$ とその他のバースマーク情報の類似量を示す．類似量が 1 に近いほどソフトウェア間の類似性が高い．

また，DB_KBSB，DB_KBSB_FDEL についての実験結果も同様に，ソフトウェア群 1 については**表 11**，ソフトウェア群 2 については**表 12** に示した．

### 5.6 実験結果の考察

実験結果をもとに，考察を行う．

#### 5.6.1 失敗呼び出し削除による影響

表 9 より，失敗呼び出し削除をしない場合は A と A' の類似量が小さくなるが，失敗呼び出し削除をすると，類似量が大きくなることが分かる．したがって，本実験で挿入した API 関数呼び出しが削除されていることが確認できる．

表 9，表 10，表 11，表 12 より，A' についての類似量以外では，類似量導出関数にかかわらず，E_api と E_fdel で類似量に大きな差は現れないことが分かる．

よって，失敗呼び出し削除による類似量への影響は問題にならない程度であることが確認できる．

#### 5.6.2 編集距離による類似量導出関数の性能

表 9 において，S_lms による類似量は，$C_3$ についての類似量が大きく，$B_1$，$B_2$ についての類似量が小さくなっているなど，バースマークとして機能していないことが分かる．しかし，S_ed による類似量は A・$B_i$ 間と A・$C_i$ 間を比較すると，最小でも 2 倍近くの開きがあり，バースマークとして機能していることが分かる．また，表 10 においても，$\beta_2$ 以外については，バースマークとして機能していることが分かる．よっ

**表 9** E_fdel，S_ed，S_ed2 の実験結果 1（数値は類似量）
Table 9 Experiment result of E_fdel, S_ed, S_ed2, No.1 (The numerical value is similar amount).

| | | 比較ソフトウェア | | | | | |
|---|---|---|---|---|---|---|---|
| | | A:A' | A:$B_1$ | A:$B_2$ | A:$C_1$ | A:$C_2$ | A:$C_3$ |
| バースマーク | DB_SEQ | 0.0529 | 0.0174 | 0.0443 | 0.0105 | 0.0563 | 0.1211 |
| | DB_SEQ_FDEL | 0.7296 | 0.0176 | 0.0446 | 0.0105 | 0.0264 | 0.1290 |
| | DB_SEQ_ED | 0.2293 | 0.6927 | 0.5752 | 0.2306 | 0.1787 | 0.3037 |
| | DB_SEQ_FDEL_ED | 0.9925 | 0.6933 | 0.5765 | 0.2275 | 0.1762 | 0.3169 |
| | DB_SEQ_ED2 | 0.9919 | 0.9253 | 0.8003 | 0.2761 | 0.5897 | 0.3966 |
| | DB_SEQ_FDEL_ED2 | 0.9958 | 0.9249 | 0.7993 | 0.2662 | 0.5847 | 0.3729 |

**表 10** E_fdel，S_ed，S_ed2 の実験結果 2（数値は類似量）
Table 10 Experiment result of E_fdel, S_ed, S_ed2, No.2 (The numerical value is similar amount).

| | | 比較ソフトウェア | | | | | |
|---|---|---|---|---|---|---|---|
| | | $\alpha:\beta_1$ | $\alpha:\beta_2$ | $\alpha:\beta_3$ | $\alpha:\gamma_1$ | $\alpha:\gamma_2$ | $\alpha:\gamma_3$ |
| バースマーク | DB_SEQ | 0.7098 | 0.0398 | 0.5833 | 0.0252 | 0.0237 | 0.4862 |
| | DB_SEQ_FDEL | 0.7131 | 0.0401 | 0.6176 | 0.0258 | 0.0236 | 0.5183 |
| | DB_SEQ_ED | 0.9389 | 0.1907 | 0.9910 | 0.3180 | 0.2089 | 0.5833 |
| | DB_SEQ_FDEL_ED | 0.9379 | 0.1877 | 0.9910 | 0.3174 | 0.2062 | 0.5854 |
| | DB_SEQ_ED2 | 0.9984 | 0.5299 | 0.9980 | 0.4675 | 0.3673 | 0.8110 |
| | DB_SEQ_FDEL_ED2 | 0.9984 | 0.5281 | 0.9982 | 0.5041 | 0.3706 | 0.8404 |

**表 11** DB_KBSB，DB_KBSB_FDEL の実験結果 1（数値は類似量）
Table 11 Experiment result of DB_KBSB, DB_KBSB_FDEL, No.1 (The numerical value is similar amount).

| | k 値 | 比較ソフトウェア | | | | | |
|---|---|---|---|---|---|---|---|
| | | A:A' | A:$B_1$ | A:$B_2$ | A:$C_1$ | A:$C_2$ | A:$C_3$ |
| DB_KBSB | 1 | 1.0000 | 0.9916 | 0.9748 | 0.6975 | 0.7731 | 0.9832 |
| | 2 | 0.8879 | 0.8214 | 0.8144 | 0.5184 | 0.4851 | 0.8354 |
| | 3 | 0.8473 | 0.7626 | 0.7361 | 0.4704 | 0.3919 | 0.7343 |
| | 4 | 0.8194 | 0.6982 | 0.6453 | 0.4282 | 0.3300 | 0.6565 |
| | 5 | 0.7989 | 0.6509 | 0.5780 | 0.3972 | 0.2978 | 0.6005 |
| | 6 | 0.7812 | 0.6083 | 0.5236 | 0.3643 | 0.2788 | 0.5494 |
| | 7 | 0.7663 | 0.5751 | 0.4806 | 0.3394 | 0.2658 | 0.5133 |
| | 8 | 0.7514 | 0.5462 | 0.4447 | 0.3211 | 0.2560 | 0.4808 |
| | 9 | 0.7398 | 0.5205 | 0.4173 | 0.3057 | 0.2463 | 0.4522 |
| DB_KBSB_FDEL | 1 | 1.0000 | 0.9913 | 0.9565 | 0.6870 | 0.7652 | 0.9826 |
| | 2 | 0.9875 | 0.8154 | 0.7849 | 0.5197 | 0.4767 | 0.8351 |
| | 3 | 0.9802 | 0.7579 | 0.7075 | 0.4698 | 0.3843 | 0.7336 |
| | 4 | 0.9729 | 0.6918 | 0.6176 | 0.4276 | 0.3233 | 0.6550 |
| | 5 | 0.9681 | 0.6462 | 0.5538 | 0.3949 | 0.2938 | 0.5982 |
| | 6 | 0.9616 | 0.6050 | 0.5009 | 0.3608 | 0.2765 | 0.5469 |
| | 7 | 0.9553 | 0.5719 | 0.4582 | 0.3354 | 0.2628 | 0.5105 |
| | 8 | 0.9479 | 0.5428 | 0.4230 | 0.3158 | 0.2511 | 0.4775 |
| | 9 | 0.9414 | 0.5171 | 0.3962 | 0.2999 | 0.2405 | 0.4486 |

**表 12** DB_KBSB，DB_KBSB_FDEL の実験結果 2（数値は類似量）
Table 12 Experiment result of DB_KBSB, DB_KBSB_FDEL, No.2 (The numerical value is similar amount).

| | k 値 | 比較ソフトウェア | | | | | |
|---|---|---|---|---|---|---|---|
| | | $\alpha:\beta_1$ | $\alpha:\beta_2$ | $\alpha:\beta_3$ | $\alpha:\gamma_1$ | $\alpha:\gamma_2$ | $\alpha:\gamma_3$ |
| DB_KBSB | 1 | 1.0000 | 0.7901 | 1.0000 | 0.7901 | 0.8395 | 0.9012 |
| | 2 | 0.9971 | 0.6589 | 0.9650 | 0.5860 | 0.5977 | 0.7726 |
| | 3 | 0.9885 | 0.6236 | 0.9497 | 0.4813 | 0.4799 | 0.7069 |
| | 4 | 0.9841 | 0.5978 | 0.9439 | 0.4032 | 0.3948 | 0.6455 |
| | 5 | 0.9802 | 0.5843 | 0.9407 | 0.3761 | 0.3578 | 0.6246 |
| | 6 | 0.9756 | 0.5714 | 0.9384 | 0.3566 | 0.3345 | 0.6092 |
| | 7 | 0.9729 | 0.5654 | 0.9308 | 0.3464 | 0.3258 | 0.6025 |
| | 8 | 0.9700 | 0.5541 | 0.9195 | 0.3386 | 0.3162 | 0.5948 |
| | 9 | 0.9674 | 0.5437 | 0.9147 | 0.3306 | 0.3096 | 0.5894 |
| DB_KBSB_FDEL | 1 | 1.0000 | 0.7922 | 1.0000 | 0.7792 | 0.8442 | 0.8961 |
| | 2 | 0.9970 | 0.6568 | 0.9645 | 0.5769 | 0.5917 | 0.7692 |
| | 3 | 0.9884 | 0.6215 | 0.9505 | 0.4745 | 0.4760 | 0.7016 |
| | 4 | 0.9829 | 0.5930 | 0.9431 | 0.3966 | 0.3971 | 0.6414 |
| | 5 | 0.9792 | 0.5800 | 0.9411 | 0.3683 | 0.3475 | 0.6195 |
| | 6 | 0.9746 | 0.5670 | 0.9398 | 0.3473 | 0.3231 | 0.6043 |
| | 7 | 0.9715 | 0.5612 | 0.9327 | 0.3374 | 0.3145 | 0.5968 |
| | 8 | 0.9686 | 0.5498 | 0.9226 | 0.3286 | 0.3041 | 0.5904 |
| | 9 | 0.9664 | 0.5388 | 0.9185 | 0.3200 | 0.2965 | 0.5855 |

て，本実験においては，S_ed は S_lms よりも対象ソフトウェアの機能の僅かな違いによる影響が抑えられているといえる．

S_ed2 による類似量は，表 9 より，API 関数挿入攻撃の影響を打ち消すことに成功していることが分かる．さらに $B_i$ についての類似量も 1 に近くなっており，機能改変攻撃耐性も S_ed より向上していると考えられる．表 10 においても，$\beta_i$ についての類似量が全体的に向上していることが分かる．しかし，4.2 節で述べたとおり，S_ed に比べ，全体的に類似量が大き

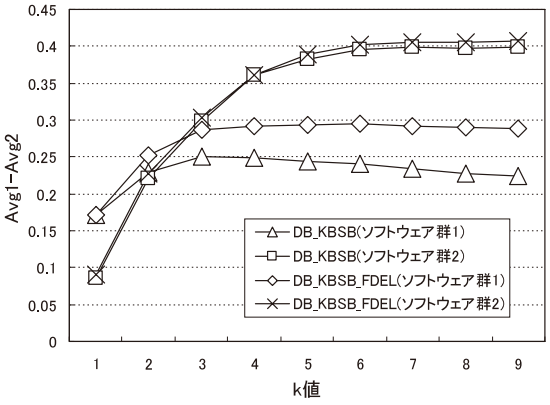


**図 3** $k$ 値の評価（平均値の差）
Fig. 3 Evaluation of $k$ (Difference of average value).

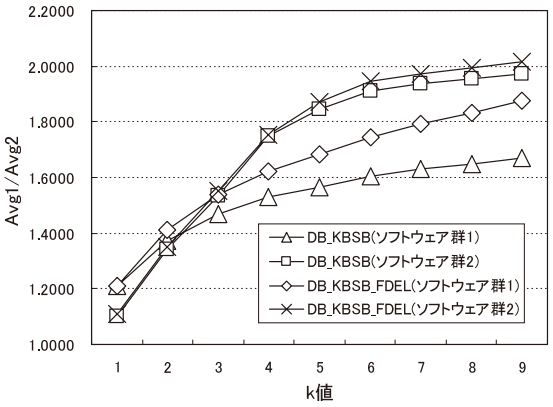


**図 4** $k$ 値の評価（平均値の比率）
Fig. 4 Evaluation of $k$ (Ratio of average value).

くなっている．特に，$C_2$ についての類似量や $\gamma_3$ についての類似量などが大きくなっており，弁別性は低下していることが確認できる．

なお，本実験結果から，S_ed および S_ed2 は S_lms よりも Sim として良好な性質を持っていると考えられるが，S_ed および S_ed2 は API 関数呼び出し履歴の取得範囲のずれが類似量に影響を与えるなどの短所も存在するため，それぞれの使い分けが必要となる．

#### 5.6.3 DB_KBSB の性能

表 11 を見ると，$C_3$ についての類似量が $B_1$，$B_2$ についての類似量と同等に大きくなっている．これより，DB_KBSB は弁別性に欠けると考えられる．しかし，A' についての類似量は大きくなっており，API 関数挿入攻撃による影響は抑えられていることが確認できるため，DB_KBSB の利用価値は十分にあると考えられる．

ここで，バースマークとして優れた性能を示す $k$ 値を求める．優れた性能を示すとは，攻撃耐性と弁別性に優れていることである．すなわち，A，$\alpha$ と派生物（A'，$B_i$，$\beta_i$）の類似量が大きく，非派生物（$C_i$，$\gamma_i$）の類似量が小さくなる傾向にある方が良い．

この評価を行うために，本論文では，派生物についての類似量の平均値（Avg1）と，非派生物についての類似量の平均値（Avg2）から，それらの差（Avg1 − Avg2）および比率（Avg1/Avg2）をソフトウェア群ごとに求める．これらの値が大きいほど，バースマークとしての性能に優れた傾向にあるといえる．これらの結果を，**図 3**，**図 4** に示した．

図 3，図 4 より，ソフトウェア群 1 については $k$ 値が 1〜3 程度のときに差，比率の変化が大きい傾向にあり，ソフトウェア群 2 については $k$ 値が 1〜5 程度のときに差，比率の変化が大きい傾向にある．また，今回の実験範囲では，$k$ 値が大きくなっても差，比率が大きく下がることはなかった．したがって，本評価により，DB_KBSB，DB_KBSB_FDEL ともに，$k$ 値としては 5 程度以上を目安に用いるべきであるといえる．

#### 5.6.4 $\beta_2$ についての考察

ここで $\beta_2$ について述べる．$\beta_2$ は $\alpha$ の派生物であるにもかかわらず全バースマークにおいて類似量が小さくなっている．$\beta_2$ の API 関数呼び出し履歴を見ると，ある特定の API 関数が非常に多くの回数呼び出されていた．この API 関数は他ソフトウェアの呼び出し履歴でも最も多く記録されているが，呼び出し回数が膨大であるために履歴取得対象としなかった 4 関数ほど多くはなかった．そのため，本実験では履歴取得対象としたが，$\beta_2$ では呼び出し回数が突出していたため，その影響で類似量が小さくなったと考えられる．

## 6. まとめと今後の課題

本論文のまとめと，今後の課題について述べる．

### 6.1 ま と め

本論文では，API 関数呼び出し履歴を用いた動的バースマークに対する攻撃方法の整理を行い，既存バースマークでは耐性のない攻撃方法に着目した．そこで，バースマークは複数の関数から構成されているととらえたうえで，攻撃耐性の向上を考慮したバースマーク，バースマーク関数を提案した．そして評価実験を行い，各提案バースマークの有効性を確認することができた．

### 6.2 今後の課題

今後の課題について項目別に述べる．

#### 6.2.1 失敗呼び出し以外の無効呼び出し検知方法

失敗呼び出し削除はあくまでも盗用者の攻撃手段を

減らすというだけであり，他の無効呼び出しを用いた攻撃には耐性を持たない．そして，4.1 節で述べた打消呼び出し，正常無効呼び出しの検知方法については，本論文では提案していない．よって，打消呼び出し，正常無効呼び出しの検知方法を模索する必要がある．また，未知の無効呼び出しに関する調査も行い，その検知方法を模索することも必要である．

#### 6.2.2 具体的な Cmp の提案

2.3 節で提案したバースマーク構成のうち，Cmp は一般性を考慮して組み込んだものであり，本論文では具体的な関数を提案していない．Cmp の案としては，片方の抽出情報に含まれる種類の抽出成分以外は削除する，などがあげられる．このような Cmp を用いて有用なバースマークが構成できないかの検討も課題としてあげられる．

#### 6.2.3 未実験のバースマーク

DB_REP や DB_INTV の Sim を S_ed に変えて試すなどは本論文では行っていない．このように，未実験の組合せを実験し，バースマークとしての特性を考察することも課題としてあげられる．

#### 6.2.4 バースマーク性能の定量的評価方法

本論文では，DB_KBSB において適切な $k$ 値を求めるために，類似量の平均値の差，比率を用いて評価を行った．この手法は他のバースマークにおいても適用可能であると考えられるが，類似量のばらつきの大小など，多くの評価軸があると考えられるため，特性の異なるバースマークどうしを比較することは本論文では見送った．今後，バースマークの性能を測るための評価軸を洗い出し，測定する方法を模索することが必要である．

#### 6.2.5 その他の課題

その他の課題としては，履歴取得対象とする API 関数の選択方法の明確な基準の策定，異なる版の Windows を実験に用いたときの影響の調査などがあげられる．

**謝辞** 四方順司准教授には，提案方式の評価に関して有益なコメントをいただいた．ここに記して感謝する．

## 参 考 文 献

1) Collberg, C. and Thomborson, C.: Watermarking, Tamper-proofing, and Obfuscation — Tools for software protection, *IEEE Trans. Softw. Eng.*, Vol.28, No.8, pp.735–746 (2002).
2) 古田壮宏，真野芳久：実行系列の抽象表現を利用した動的バースマーク，電子情報通信学会論文誌 D-I，Vol.J88-D1, No.10, pp.1595–1598 (2005).
3) Gilleland, M.: Levenshtein Distance (online). Available from <http://www.merriampark.com/ld.htm> (accessed 2006-11-21).
4) Grover, D.: The protection of computer software: Its technology and applications, *The British Computer Society Monographs in Informatics*, 2nd Edition, pp.142–144, Cambridge University Press (1992).
5) はせた：haseta2003's Homepage (online). Available from <http://haseta2003.hp.infoseek.co.jp/> (accessed 2006-8-12).
6) はせた：はせたです (online). Available from <http://hp.vector.co.jp/authors/VA012911/> (accessed 2006-8-12).
7) 林晃一郎，楓 基靖，真野芳久：特徴抽出と抽象化による動的バースマークの構成とその検証，情報処理学会研究報告コンピュータセキュリティ，No.2005-CSEC-031, pp.31–36 (Dec. 2005).
8) Heidenreich, F.: Mp3tag — der universelle Tag Editor (online). Available from <http://www.mp3tag.de/> (accessed 2006-8-12).
9) 委員長，2 ちゃんねる専用ブラウザ「A Bone」(online). Available from <http://www.a-bone.net/> (accessed 2006-8-12).
10) インプレスジャパン：Sigma Designs，RMP4 に XVID からのコード流用を認める (online). Available from <http://www.watch.impress.co.jp/av/docs/20020828/sigma.htm> (accessed 2006-8-27).
11) ITmedia: News: GPL 違反指摘の「DVD コンバータ」，ソースコード開示「CSS 解除機能は搭載していない」(online). Available from <http://www.itmedia.co.jp/news/0302/13/njbt_03.html> (accessed 2006-8-27).
12) Larus, J.R.: Whole Program Paths, *SIGPLAN '99 Conference on Programming Language Design and Implementation* (May 1999).
13) Kobarin：Kobarin のホームページ (online). Available from <http://home7.highway.ne.jp/Kobarin/> (accessed 2006-8-12).
14) 黒田 徹：Teatime (online). Available from <http://www.vector.co.jp/soft/win95/art/se131175.html> (accessed 2006-8-12).
15) MERCURY: MERCURY's Software Lab. (online). Available from <http://www5.wisnet.ne.jp/~mercury/> (accessed 2006-8-12).
16) Microsoft Corporation: Windows® Server 2003 SP1 Platform SDK Web Install (online). Available from <http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en> (accessed 2006-11-21).
17) ギコナビプロジェクト：ギコナビ (online). Available from <http://gikonavi.sourceforge.

jp/> (accessed 2006-8-12).
18) Myles, G. and Collberg, C.: Detecting Software Theft via Whole Program Path Birthmarks, *Information Security Conference*, pp.404–415 (Sep. 2004).
19) Myles, G. and Collberg, C.: k-gram Based Software Birthmarks, *ACM 2005 Symposium on Applied Computing*, pp.314–318 (Mar. 2005).
20) 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 D，Vol.J89-D，No.8，pp.1751–1763 (Aug. 2006).
21) Open Jane プロジェクト：CVS Repository — directory — SourceForge:jane (online). Available from <http://cvs.sourceforge.jp/cgi-bin/viewcvs.cgi/jane/> (accessed 2006-8-18).
22) おりび：えまのんほーむ (online). Available from <http://www.emanong.net> (accessed 2006-8-12).
23) P'sSoft: P'sSoft (online). Available from <http://www.cam.hi-ho.ne.jp/akuma/main.html> (accessed 2006-8-12).
24) ◆Style/kK.s: 2 ちゃんねる専用ブラウザ「Jane Style」(online). Available from <http://janestyle.s11.xrea.com/> (accessed 2006-8-12).
25) Tamada, H., Nakamura, M., Monden, A. and Matsumoto, K.: Java Birthmarks — Detecting the Software Theft, *IEICE Trans. Inf. and Syst.*, Vol.E88-D, No.9, pp.2148–2158 (2005).
26) View: Jane View (online). Available from <http://www.geocities.jp/jview2000/> (accessed 2006-8-12).

(平成 18 年 11 月 27 日受付)
(平成 19 年 6 月 5 日採録)

**森山　修**

2006 年横浜国立大学工学部電子情報工学科卒業．同年同大学大学院環境情報学府情報メディア環境学専攻博士課程前期に進学，現在に至る．ソフトウェアセキュリティの研究に従事．

**古江　岳大**

2002 年横浜国立大学工学部電子情報工学科卒業．2004 年同大学大学院環境情報学府情報メディア環境学専攻博士課程前期修了．現在，同大学院同専攻博士課程後期に在学中．情報セキュリティの研究に従事．

**遠山　毅**

2004 年横浜国立大学工学部電子情報工学科卒業．2006 年同大学大学院環境情報学府情報メディア環境学専攻博士課程前期修了．現在，同大学院同専攻博士課程後期に在学中．情報ハイディングに興味を持つ．

**松本　勉**（正会員）

情報セキュリティ，暗号，耐タンパ性，バイオメトリクス，人工物メトリクス，その他，情報学全般を専門とする．東京大学大学院工学系研究科電子工学専攻博士課程修了，工学博士（1986 年 3 月）．1986 年 4 月より横浜国立大学工学部に専任講師として勤務．同助教授，同教授を経て，2001 年 4 月より同大学大学院環境情報研究院教授．2007 年 4 月より社会環境と情報部門長を兼任．この間，日本銀行金融研究所客員研究員，カールスルーエ大学（ドイツ）客員教授，ケンブリッジ大学（イギリス）研究員，等を兼務．現在，国際暗号学会（IACR）理事．日本学術会議連携会員．CRYPTREC 暗号技術検討会構成員．暗号モジュール試験および認証制度（JCMVP）における技術審議委員会委員長．ISO/TC68（金融サービス）国内委員会委員長．電子情報通信学会平成 6 年度業績賞「情報セキュリティの基礎技術に関する先駆的研究」，2006 年第 5 回ドコモ・モバイル・サイエンス賞・先端技術部門・優秀賞等を受賞．