Java対応ランタイムデータ捕捉ソフトウェア

松本 勉[†] 赤井 健一郎[†] 中村 豪一[††]
大内 功[†††] 竹脇 和也[†††] 村瀬 一郎[††]

Java の仮想機械上で動くプログラムの実行過程で現れるランタイムデータを捕捉するソフトウェア（DataExtractor）について研究し，Java™ Platform Debugger Architecture を拡張する方法により，その 1 つを開発した．本論文はその詳細を記述するものである．DataExtractor はランタイムデータを解析してプログラムの耐タンパー性を評価する際の強力な道具として有用である．

Runtime Data Capturing Software for Java

Tsutomu Matsumoto,[†] Kenichiro Akai,[†] Goichi Nakamura,[††]
Kou Ouchi,[†††] Kazuya Takewaki[†††] and Ichiro Murase[††]

We have studied a kind of software tool that can capture all runtime data while a Java program is running. We call it DataExtractor. We have implemented a form of the DataExtractor by expanding Java™ Platform Debugger Architecture. In this paper, we explain the implementation details of the DataExtractor. The DataExtractor is useful as a powerful tool for evaluating tamper resistance of a program by analyzing runtime data of it.

1. はじめに

様々なプラットフォーム上で Java 言語の有用性が増している．筆者らは，Java の仮想機械上（JVM）で動くプログラムの実行過程で現れるランタイムデータを捕捉するソフトウェア（DataExtractor）について研究している[1)]．DataExtractor は，プログラムの計算途中の中間値も捕捉可能であることを特徴とする．DataExtractor は，ランタイムデータを解析してプログラムの耐タンパー性を評価する際の強力な道具として有用である[2)]．

Java 環境においては，Java™ Platform Debugger Architecture（JPDA）[3)] が Java デバッガ開発用パッケージとして提供されているが，そのままでは DataExtractor としては用いることはできない．本論文は，この JPDA をある意味で拡張するというアプローチにより DataExtractor を構成する具体的方法を提案するものであり，この方法に従って実装した DataExtractor の詳細を記述するものである．本論文の構成は以下のとおりである．まず 2 章でランタイムデータ探索型耐タンパー性評価法についてまとめ，3 章では JVM と JPDA に関する事項を述べる．そして 4 章において DataExtractor の構成方法と実装結果を示し，5 章でまとめと今後の課題について述べる．

† 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
†† 株式会社三菱総合研究所
Mitsubishi Research Institute, Inc.
††† エム・アール・アイシステムズ株式会社
MRI Systems, Inc.

2. ランタイムデータ探索型耐タンパー性評価法

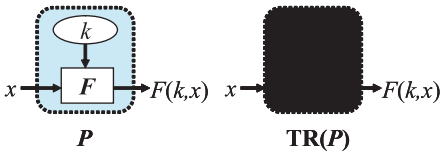
2.1 耐タンパーソフトウェア

耐タンパー性とは，ソフトウェアまたはハードウェアのモジュール内部に存在する秘密情報の観測や，モジュールが実現する機能の改変が困難な性質のことである．耐タンパー性を構成する性質として，秘密データ観測が困難な性質を秘密データ守秘性，機能を改変することが困難な性質を機能改変困難性と呼ぶことにする．特に耐タンパー性を有するソフトウェアを耐タンパーソフトウェアと呼ぶ．

耐タンパーソフトウェア技術として，解析が困難となるように保護対象プログラムを機能的に等価に変換するオブフスケーション技術があり，広く検討されている．

2.2 評価モデル

ある暗号関数を実装し内部に鍵 k を持つ耐タンパー



k : 秘密データ（鍵）
F : 暗号関数
x : 暗号関数への入力
$F(k,x)$: 暗号関数からの出力

図 1 評価対象プログラムのタイプ
Fig. 1 Type of target program.

化対象プログラム P に対し，鍵 k の導出が困難となるようにソフトウェア耐タンパー化技術 TR を適用してできた評価対象プログラム TR(P) が与えられたとする（**図 1**）．筆者らの目的は評価対象プログラム TR(P) を解析して，内部に隠された k を導出することである．このとき，P が実装する暗号関数は評価者（解析者）に与えられているものとする．

評価対象プログラム TR(P) を入手した解析者は，内部に隠された鍵 k を導出する解析を行う．その解析には，入出力解析と内部解析が考えられる．入出力解析は，TR(P) の入出力値の関係から k の導出を行うもので，通常の暗号解読に相当する．他方，内部解析は，入出力解析に加えて，TR(P) の記述や実行時のデータフロー等の内部の情報を利用して k の導出を行うものである．それぞれの解析により，鍵 k の導出にかかるコストの最小値を $Cost_{io}^{min}$，$Cost_{inside}^{min}$ とする．内部解析は外部解析に比して，利用できる情報が多いため，次式が成り立つ．

$$Cost_{inside}^{min} \leq Cost_{io}^{min}.$$

この式において，あるが等号を満たす場合は，内部解析からは十分な情報を得ることができなかったわけであり，TR(P) は十分な秘密データ守秘性を有していると見なすことができる．また，この式は，$Cost_{io}^{min}$ と $Cost_{inside}^{min}$ を比較することにより，TR(P) が有する秘密データ守秘性を評価可能であることを示唆している．

2.3 評価法の概略

評価対象プログラム TR(P) の記述において巧妙に鍵 k が秘匿されていたとしても，実行時にメモリ上に k の実体が現れることがある．このような TR(P) は，メモリ上から k を取り出されるという脆弱性を有していると考えられる．

そこで，本評価法は評価対象プログラム TR(P) の実行過程にメモリ上に現れる全データを抽出して秘密データの候補集合 Ω とし，Ω の元を利用して鍵 k を導出する．鍵 k の導出には，P が実装している暗号関数の知識を利用する．そのため，**図 2** のように DataExtractor と Checker を組み合わせることで，本評価法を実現する．DataExtractor は，TR(P) の実行を監視して，メモリ上からデータを取り出していき，秘密データの候補集合 Ω を作成する．Checker は，仮定より解析者には開示されている暗号関数の性質を用いて，Ω の元の間の関係から鍵 k を導出すると同時に，導出に要した時間等を計測し，これらを結果として出力するソフトウェアである．

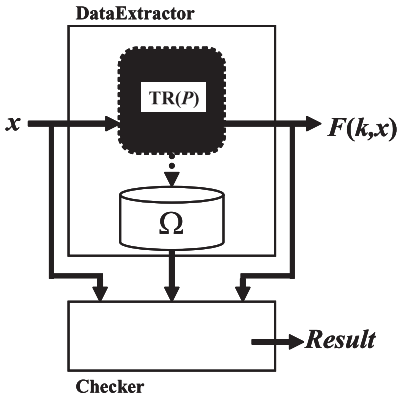


図 2 ランタイムデータ探索型耐タンパー性評価法
Fig. 2 Evaluating tamper resistance by searching runtime data.

本評価法は，すべての過程を計算機上で実装可能である．したがって，もしある評価対象プログラム TR(P) に対して本評価法が適用できたとすると，十分な知識を持たないような解析者であっても鍵 k の導出が可能となることを示している．このことから，本評価法を用いて鍵 k の導出が短時間で行うことができるような評価対象プログラム TR(P) は，秘密データの守秘性の点で脆弱であると判断可能である．しかし，本評価法を用いて短時間で k の導出ができない TR(P) があったとしても，十分な秘密データの守秘性を有しているとは判断できない．なぜならば，他の解析方法を用いた方が，本評価法を用いるよりも短時間で鍵 k の導出が可能となる場合があるからである．以上より，秘密データ守秘性の点から，本評価法に対して十分な強度を持つことは，耐タンパーソフトウェアを構成するうえでの最低限の条件であるということができる．

3. Java 環境

3.1 Java

筆者らは，Java を対象に耐タンパーソフトウェア

の研究を行っている．Java は，プラットフォーム非依存で，ネットワークとの親和性が高いことを特徴とする言語である．Java は，Web ブラウザで利用される Java アプレットをはじめとして，携帯電話での利用も進んでおり，プログラム言語として確固とした地位を築いたといえる．他方，プラットフォーム非依存の実現により，Java クラスファイルはあらゆるプラットフォームで共通となっている．また，Java クラスファイルを構成するバイトコードもオブジェクト指向で，シンプルで理解しやすい構造になっている．そのため，バイナリで記述されている Java クラスファイルをニーモニックコードに変換する逆アセンブラや，ソースコードレベルの記述に変換しなおす逆コンパイラが開発されており，十分な性能を有しているものが多い．そのため，これらの解析ソフトウェア対策として，逆コンパイラの阻止や逆コンパイラが使用されたとしても解析が困難になるようにプログラムの等価変換を行うオブフスケータと呼ばれるソフトウェアの研究・開発が行われている．筆者らは，このような状況を鑑み，Java は耐タンパーソフトウェアの研究を進めるうえで適した言語であると考え，これを研究対象の言語としている．

サンマイクロシステムズが提供する Java2 SDK では，Java で記述されたソースコードは，javac コンパイラを用いて，バイトコードで記述された Java クラスファイルに変換される．Java クラスファイルは Java Virtual Machine（JVM）と呼ばれる仮想機械上で実行される．したがって，ランタイムデータ探索型耐タンパー性評価法を構成する DataExtractor は，JVM の動作を直接監視し，メモリ上に現れるデータを抽出していく機能が求められる．

DataExtractor と同じような機能を提供するソフトウェアとして，デバッガが存在する．デバッガは開発支援ソフトウェアであり，開発中のプログラムに内在するバグを発見する際に使用する．サンマイクロシステムズは，Java デバッガ jdb を提供している．また，jdb の開発を行うための Java™ Platform Debugger Architecture（JPDA）を提供している．しかし，後述する理由により，JPDA のみでは DataExtractor の機能を実現することはできないと考えた．そこで，JPDA に手を加えることで，DataExtractor の機能を実現した．

3.2 Java クラスファイルの構造

Java クラスファイルは，そのクラスの性質を定めるフィールドと，C 言語の関数に相当するメソッドにより構成される．メソッドにはバイトコードの組合せが記述されており，それによりメソッドの機能が決定される．バイトコードは，JVM 上で使用可能な命令を 1 バイトで表現したもので，その仕様は JVM の構成に沿って，厳格に定められている．例として，**表 1** にバイトコードのいくつかを示す．

表 1 バイトコードの例
Table 1 Bytecode examples.

バイトコード	命令	仕様
0x10	bipush	1 バイトの値をオペランドスタックに積む
0x12	ldc	1 ワードの値をオペランドスタックに積む
0x15	iload	ローカル変数の値をオペランドスタックに積む
0x36	istore	オペランドスタックから 1 ワードをローカル変数に格納する
0x61	iadd	オペランドスタックに積まれた 2 ワードの値を足す

1 ワードは 32 ビットの値を示す．

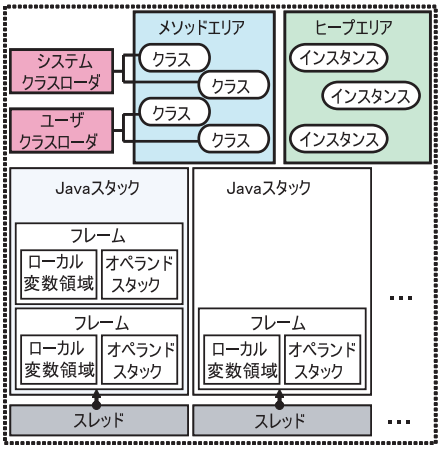


図 3 DataExtractor の構成
Fig. 3 The architecture of DataExtractor.

3.3 JVM

実行環境である JVM は，Java クラスファイル中に記述されたバイトコードを逐一解釈実行するインタプリタマシンである．その内部構造は，ロードした Java クラスファイルを記憶するメソッドエリア，Java クラスファイルのインスタンスならびに配列を記憶しておくヒープエリア，スレッドごとに割り当てられプログラムの実行状態を記憶する Java スタックにより構成されている．Java スタックには，メソッドの実行状態を記録しているフレームが積まれていく．1 つのフレームは，ローカル変数領域と，オペランドスタックにより構成されている（**図 3**）[4]．

実行される Java クラスファイルは，クラスローダを通してメソッドエリアにロードされる．Java クラ

スファイルに記述されたメソッドが実行される際に，1 つのフレームが JVM 内の Java スタックに積まれる．そして，その実行は，メソッドに記述されているバイトコードに従って実行される．演算は，フレーム内のオペランドスタックに値を積みながら行われる．ローカル変数領域は，オペランドスタックで行われた演算結果等をストアしておくために用いられる．

なお，JVM の仕様では，配列データはたとえプログラムの記述上でメソッド内にしか現れない場合でも，その実体はフレーム内のローカル変数やオペランドスタックには現れず，その参照のみが現れる．配列データの実体は，ヒープエリアに格納される．

3.4 JPDA

Java デバッガ開発用に提供された Java^{TM} Platform Debugger Architecture（JPDA）は，Java Virtual Machine Debugger Interface（JVMDI），Java Debug Wire Protocol（JDWP），Java Debugger Interface（JDI）の 3 つで構成される．JVMDI は，JVM のネイティブインタフェースで，C で記述されており，デバッグに必要なサービスを定義する．JDWP は，デバック中のプロセスへの要求や情報の仕様を規定したプロトコルである．JDI は，Pure Java で記述された Java インタフェースである．

JPDA を用いて実装されるデバッガでは，Java クラスファイルにデバック情報を付加するためのデバックオプション “-g” をつけてコンパイルを行う必要がある．“-g” オプションなしでコンパイルした Java クラスファイルを JDI から操作しようとすると，ローカル変数にアクセスすることができない．また，“-g” オプションつきでコンパイルをしたとしても，ヒープエリアやオペランドスタックに直接アクセスするインタフェースは，Java2 SDK 1.3 では提供されていない．

4. DataExtractor の構築

4.1 設計方針

DataExtractor の機能は，Java クラスファイルの実行中にメモリ上に現れたデータを抽出していき，秘密データの候補集合 Ω を生成することである．3.2 節で示したように，Java クラスファイルの実行中のメモリデータは，JVM のヒープエリア，フレーム内のローカル変数領域ならびにオペランドスタックに現れる．また，評価対象プログラム TR(P) は，デバックオプション “-g” をつけないでコンパイルされていると考えられる．というのは，“-g” オプションつきでコンパイルされている Java クラスファイルは，解析において有益な情報が付属しているからであり，耐タンパー性を高めることを考えた場合にはそれをつけないでコンパイルした方が有利であると考えられるからである．したがって，DataExtractor には “-g” オプションなしでコンパイルされた Java クラスファイルであってもヒープエリアやオペランドスタックにアクセスする能力が求められる．

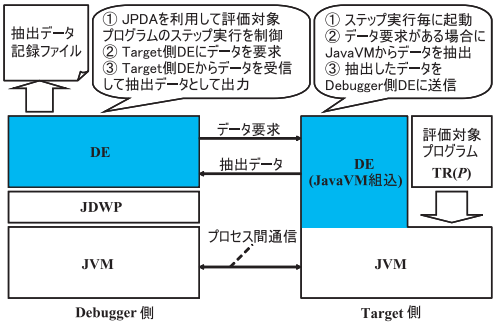


図 4 DataExtractor の構成
Fig. 4 The architecture of DataExtractor.

Java 対応 DataExtractor を実装する場合，3.4 節で示した JPDA のみで実装を行うことと，JVM を拡張して実装することとがまず考えられる．しかし，JPDA にはオペランドスタックやヒープエリアに直接アクセスできるようなインタフェースがないため，JPDA のみで DataExtractor を実現することはできない．また，JVM の拡張による DataExtractor の実現では，作業面での負荷が大きいことが予想された．そこで，JPDA を拡張することで作業面の負荷を減らしつつ，DataExtractor の機能を実現することとした．

4.2 実装の全体像

上記設計方針に従い DataExtractor を**図 4** のように構成した．環境は，Windows2000，Java2 SDK1.3 とし，この上で動作するように DataExtractor を構成した．

DataExtractor（以下，DE）は，Target 側と Debugger 側の 2 つに分かれており，これらが並行して動作することにより機能する（図 4）．Target 側 DE は評価対象プログラムを実行する JVM に組み込まれ，オペランドスタックやヒープエリアにアクセスする．Debugger 側 DE は JPDA を利用して，評価対象プログラムを実行する JVM を制御すると同時に，実行されたバイトコードを解析して，データ抽出要求を Target 側 DE に出す．Target 側 DE は JVM が評価対象プログラムをステップ実行するごとに，JVM のオペランドスタック等にアクセスして，抽出の対象となるデータを補足して，共有メモリまたは共有ファイルを介して Debugger 側 DE に受け渡す．Debugger

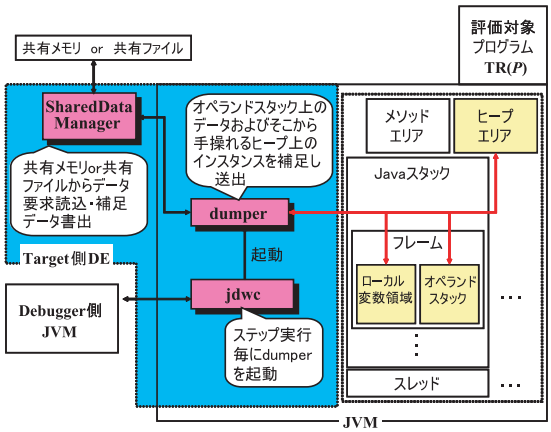


図 5 Target 側 DataExtractor の構成
Fig. 5 The architecture of Target-side DataExtractor.

側 DE は，受け取ったデータを整形し，ファイルに書き出す．

Target 側 DE の構成を**図 5** に示す．JVM には，C 言語で記述された jdwp 関数が標準で組み込まれている．jdwp 関数は，JVM のステップ実行を制御している．そこで，dumper 関数を別途用意し，jdwp 関数が呼び出されると dumper 関数を呼び出すように jdwp 関数を拡張した．dumper 関数は，JVM 内にアクセスして，データ要求が Target 側 DE から出されている場合には，そのデータを取り出してくる．また，C 言語で SharedDataManager 関数を作成し，これが dumper によって取り出されたデータを共有メモリまたは共有ファイルに書き出すことで，Debugger 側 DE に受け渡しを行うようにした．

3.3 節で示したように，JVM は Java クラスファイルのバイトコードを逐次解釈実行していく．Java スタックもオペランドスタックもスタック構造であるので，バイトコードの命令を実行した直後のデータはスタックの一番上に詰まれることになる．そこで，実行過程のすべてのデータを抽出するためには，バイトコード実行直後に Java スタックの一番上に積まれたフレーム内のオペランドスタックにおいて，その一番上に積まれたデータを抽出することとした．しかし，JVM の仕様では，配列はその参照のみがオペランドスタックに積まれる．そこで，配列を抽出する際には，オペランドスタックに積まれている参照から，ヒープエリア内にある配列の実体を特定し，それを抽出するようにした．

4.3 実行の様子

実装した DataExtractor を実行した結果を示す．実行したプログラムは TargetProgram1，TargetProgram2 の 2 つを用いた．それぞれのソースプログラムと，DataExtractor で実行過程のデータを抽出した結果を**図 6**，**図 7** に示す．図 6，図 7 とも上段がソースプログラム，下段が DataExtractor による抽出結果である．

TargetProgram1 は，単純な算術演算を行うプログラムである．32 ビット int 型の 3 つの値 `0x12345678=305419896`，`0x5a4b3c2d=1514880045`，`0x6f7e8d9c=1870564764` をそれぞれ変数 `a`，`b`，`c` に代入し，$(a \times b) = 0xfc9f5318 = -5667368$（int 型は 32 ビットなので 2^{32} を法として演算が実行される．また，Java では補数で負の整数を表現する）を計算した後 `c` を足していること，つまり `-5667368+1870564764=1813897396` が実行されたこと，が図 6 から分かる．さらに，

$a \oplus b = 305419896 \oplus 1514880045 = 1216309845$

（TargetProgram1 のソースプログラムでは 9 行目に相当）が実行されたことも分かる．これにより，基本的な演算で構成可能な共通鍵ブロック暗号を実装したプログラムであっても，実装した DataExtractor で実行過程のデータを抽出することが可能であると考えられる．

TargetProgram2 は，Java 標準パッケージで用意された多バイト長データを扱う BigInteger クラスを使用し，与えられた `x`，`b`，`n` から $b^x \bmod n$ を行うプログラムである．BigInteger は，int 型（32 ビット）配列を用いて，与えられたデータを格納する．たとえば，TargetProgram2 中の `x=0x123456789abcdef0fedcba98` は，int 配列に `[0x12345678, 0x9abcdef0, 0xfedcba98]=[30541986, -1698898192, -19088744]` として保持される．他の変数 `b`，`n` に関しても同様である．また，TargetProgram2 で $b^x \bmod n$（ソースプログラムでは 9 行目の記述 `b.modPow(x,n)`）を計算した結果も，それぞれ，DataExtractor を用いて，値を抽出できていることが分かる．これにより，多バイト長演算を用いてベキ乗剰余計算を実装したプログラムであっても，実装した DataExtractor を用いてデータ抽出が行えると考えられる．

以上から，共通鍵ブロック暗号や，ベキ乗剰余演算を用いるような公開鍵暗号を実装した評価対象プログラムに，実装した DataExtractor を使用して，データの抽出ができると考えられる．

4.4 Java2 SDK のバージョンに関する問題

Java2 では HotSpot と呼ばれる技術により JVM の高速化が図られている．Java2 SDK1.3 では，通常は HotSpot が実装された JVM が使用されるように

```
 1:public class TargetProgram1{
 2:   public static void main(String argv[]){
 3:       int a = 0x12345678;
 4:       int b = 0x5a4b3c2d;
 5:       int c = 0x6f7e8d9c;
 6:       int d = (a*b)+c;
 7:       int e = a^b;
 8:       System.out.println(d);
 9:       System.out.println(e);
10:   }
11:}
```

```
#line:dump type:class type:instruction:location:value
 1:operand_stack:int:ldc:TargetProgram1.main(java.lang.String[])+0:305419896
 2:operand_stack:int:ldc:TargetProgram1.main(java.lang.String[])+3:1514880045
 3:operand_stack:int:ldc:TargetProgram1.main(java.lang.String[])+6:1870564764
 4:operand_stack:int:iload_1:TargetProgram1.main(java.lang.String[])+9:305419896
 5:operand_stack:int:iload_2:TargetProgram1.main(java.lang.String[])+10:1514880045
 6:operand_stack:int:imul:TargetProgram1.main(java.lang.String[])+11:-56667368
 7:operand_stack:int:iload_3:TargetProgram1.main(java.lang.String[])+12:1870564764
 8:operand_stack:int:iadd:TargetProgram1.main(java.lang.String[])+13:1813897396
 9:operand_stack:int:iload_1:TargetProgram1.main(java.lang.String[])+16:305419896
10:operand_stack:int:iload_2:TargetProgram1.main(java.lang.String[])+17:1514880045
11:operand_stack:int:ixor:TargetProgram1.main(java.lang.String[])+18:1216309845
12:operand_stack:int:iload:TargetProgram1.main(java.lang.String[])+24:1813897396
13:operand_stack:int:iload:TargetProgram1.main(java.lang.String[])+32:1216309845
```

図 6 TargetProgram1 と DataExtractor による実行結果
Fig. 6 TargetProgram1 and the corresponding sample result by the DataExtractor.

図 7 TargetProgram2 と DataExtractor による実行結果
Fig. 7 TargetProgram2 and the corresponding sample result by the DataExtractor.

なっているが，HotSpot 実装がなされていない JVM も選択して使用することが可能である．本論文で示した DataExtractor の実装では，Java2 SDK1.3.1 を使用し，HotSpot 実装がなされていない JVM を動作させて，データの抽出を行うように実装してある．しかし，Java2 SDK1.4 以降では，HotSpot 実装がなされた JVM のみが使用可能となっている．HotSpot 実装がなされた JVM と HotSpot 実装がなされてい

ない JVM では構造が異なるため，本論文で示した DataExtractor を HotSpot 実装がなされた JVM に直接適用してデータ抽出することはできない．したがって，HotSpot 実装がなされた JVM で DataExtractor を機能させるためには，改めて実装する必要がある．

しかし，Java クラスファイルのバイトコード仕様は，Java2 SDK1.4 と Java2 SDK1.3 では同じであるので，Java2 SDK1.4 の javac コンパイラで作成された Java クラスファイルであっても，Java2 SDK1.3 の JVM で実行可能である（ライブラリを追加しなければならない場合もある）．したがって，上位の環境で作成された Java クラスファイルであっても，本論文で示した DataExtractor でデータの抽出を行うことは可能であると考えられる．

5. ま と め

本論文では，ランタイムデータ探索型耐タンパー性評価法において，Java で実装された評価対象プログラムの実行過程のデータをすべて抽出する機能を実現する DataExtractor の実装に関して述べた．

本論文で述べた DataExtractor は，どんな Java プログラムに対しても実行過程のデータを抽出することができる．しかし，ランタイムデータ探索型耐タンパー性評価法を実現するためには，Checker の検討が不可欠である．Checker は，評価対象プログラムが実装する暗号方式や実装方式により，その構成は異なる．評価対象プログラムが共通鍵ブロック暗号を実装し，秘密データであるラウンド鍵が DataExtractor で捕捉できない場合の Checker の構成に関する検討は先行的に行われている[5),6)]．

今後の課題は，共通鍵ブロック暗号に関してはさらに詳細に Checker の検討を進めていくこと，公開鍵暗号に関しては Checker の構成方法を具体的に検討すること，そして本論文で述べた DataExtractor を用いて実際に評価対象プログラムの耐タンパー性を評価することである．

謝辞 本論文で示した DataExtractor の開発は，主として情報処理振興事業協会「情報技術開発支援事業」（平成 13 年度および平成 14 年度）採択テーマ「署名ソフトウェアに対する全数探索型耐タンパー性評価ツール」の一環として行われたものである．

参 考 文 献

1) 松本 勉，赤井健一郎，中村豪一，大内 功，村瀬一郎：Java 仮想機械ランタイムデータ抽出ツール，電子情報通信学会技術研究報告，ISEC2002-29, pp.111–116 (2002).
2) 赤井健一郎，三澤 学，松本 勉：ランタイムデータ探索型耐タンパー性評価法，情報処理学会論文誌，Vol.43, No.8, pp.2447–2457 (2002).
3) Java™ Platform Debugger Architecture. http://java.sun.com/products/jpda/
4) Mayer, J. and Dowing, T.: *Java Virtual Machine*, O'REILLY (1997). 鷲見 豊（監訳）：Java バーチャルマシン，オライリー・ジャパン (1997).
5) 甲斐文幸，赤井健一郎，松本 勉：共通鍵暗号ソフトウェアの間接的ランタイムデータ探索型耐タンパー性評価法，コンピュータセキュリティシンポジウム 2002（CSS2002）予稿集，pp.107–112 (2002).
6) 甲斐文幸，赤井健一郎，松本 勉：共通鍵暗号ソフトウェアの間接的ランタイムデータ探索型耐タンパー性評価法（2），2003 年暗号と情報セキュリティシンポジウム（SCIS2003）予稿集，pp.1059–1064 (2003).

(平成 14 年 12 月 2 日受付)
(平成 15 年 6 月 3 日採録)

松本 勉（正会員）

1986 年東京大学大学院博士課程修了，工学博士．同年横浜国立大学工学部専任講師．同助教授，教授を経て，2001 年より同大学大学院環境情報研究院教授．1981 年より暗号や情報セキュリティの研究に従事．「明るい暗号研究会」創設メンバー．現在，暗号アルゴリズム，情報利用管理，ディジタル証拠性，情報ハイディング，バイオメトリクス，人工物メトリクス，耐タンパーソフトウェア等に広く関心を持つ．国際暗号学会 IACR 理事．暗号技術検討会構成員．ASIACRYPT '96 プログラム委員長．ASIACRYPT 2000 実行委員長．電子情報通信学会より「情報セキュリティの基礎理論」への貢献に関して業績賞を受賞．

赤井健一郎

2001 年横浜国立大学大学院工学研究科人工環境システム学専攻博士課程前期修了．同年同大学院環境情報学府情報メディア環境学専攻博士課程後期に進学，現在に至る．情報セキュリティの研究に従事．

中村　豪一

1990 年東京大学工学部計数工学科卒業．1992 年同大学院修士課程修了，同年に三菱総合研究所入社，現在に至る．プログラミング言語，情報セキュリティ，数値解析の研究に従事．応用数理学会会員．

大内　　功

1970 年生．芝浦工業大学工業経営学科卒業．エム・アール・アイシステムズ株式会社システム開発部勤務．現在，プロダクトライフサイクルマネジメント分野のエンジニアとして活動中．

竹脇　和也

1958 年生．明治大学文学部史学地理学科卒業．OS，コンパイラ・アセンブラ等基本ソフトウェアの研究開発プロジェクトを手掛ける．現在，システム開発部長としてエム・アール・アイシステムズ株式会社のソフトウェア開発事業統括を担当．

村瀬　一郎（正会員）

1963 年生．1986 年株式会社三菱総合研究所入社以後，情報技術の調査研究に従事．現在は，情報セキュリティに興味を持ち，耐タンパーソフトウェア，インフォメーションハイディング，セキュアプログラミング，ネットワークセキュリティ等幅広くセキュリティの研究を行っている．電子情報通信学会会員．