コンピュータセキュリティ　18−17
(2002.　7.　18)

Java仮想機械ランタイムデータ抽出ツール

松本 勉† 赤井 健一郎† 中村 豪一‡ 大内 功≡ 村瀬 一郎‡

†横浜国立大学大学院 環境情報学府/環境情報研究院 〒240-8501 神奈川県横浜市保土ヶ谷区常盤台 79-7
‡三菱総合研究所 〒100-8141 東京都千代田区大手町 2-3-6
≡MRIシステムズ 〒104-0053 東京都中央区晴海 3 丁目 2 番 22 号

E-mail: †{tsutomu, akai}@mlab.jks.ynu.ac.jp, ‡{gon, murase}@mri.co.jp, ≡oouchi@mrisys.co.jp

あらまし 機密データを内部において扱うソフトウエアについては，その機密データの守秘性という耐タンパー性が重要である．ソフトウエアの耐タンパー性評価法として，ソフトウエアの実行過程で計算機内部に現れるデータすべてに関して機密データあるかどうかをチェックするというランタイムデータ全数探索法がある．Java クラスファイルに対して，このランタイムデータ全数探索法を適用するために，Java 仮想機械中に現れるランタイムデータを抽出するツールを開発した．Java で記述した RSA 署名ソフトウエアに対して，開発したツールを適用し，署名生成鍵をランタイムデータとして抽出する実験を行った．
キーワード 耐タンパー，Java仮想機械，署名ソフトウエア，ランタイムデータ，全数探索

Runtime Data Extraction Tool for Java Virtual Machine

Tsutomu MATSUMOTO† Kenichiro AKAI† Goichi NAKAMURA‡

Kou OUCHI≡ and Ichiro MURASE‡

†Graduate School of Environment and Information Sciences, Yokohama National University,

79-7 Tokiwadai, Hodogaya-ku, Yokohama, 240-8501 Japan
‡Mitsubishi Research Institute, 2-3-6 Otemachi, Chiyoda-ku, Tokyo, 100-8141 Japan
≡MRIsystems, 3-2-22 Harumi, Chuo-ku, Tokyo, 104-0053 Japan

E-mail: †{tsutomu, akai}@mlab.jks.ynu.ac.jp, ‡{gon, murase}@mri.co.jp, ≡oouchi@mrisys.co.jp

Abstract Tamper-resistance,which means secret data protection in this paper, is important for softwares in which secret data are stored and calculated. There is a method to evaluate tamper-resistance of a software, the runtime data exhaustive search method. We developed a tool to extract runtime data in JVM(Java Virtual Machine). This tool is an implementation for the runtime data exhaustive search method. Then, we carried out experiments to use this tool for RSA signature software written by Java to extract RSA private key.
Keyword tamper-resistance, JVM, signature software, runtime data, exhaustive search

1. はじめに

機密データを内部において扱うソフトウエアについては，その機密データの守秘性という耐タンパー性が重要である．特に，電子署名法の施行等もあり，署名生成鍵を扱う署名ソフトウェアについては高度な耐タンパー性が求められている．

情報セキュリティ技術を構成する各種の要素技術に対して，そのセキュリティの強度を評価する技術の開発が近年になって盛んに行われている．特に，暗号の評価技術については，各暗号アルゴリズムの種類と攻撃手法毎に強度評価法が確立されている．それに比べて，ソフトウエアの耐タンパー性の強度評価法に関する議論は少ない．そのため，ソフトウエアの耐タンパー性の強度評価法を確立し，評価ツールの形にまとめることが喫緊の課題となっている．

一般に，ソフトウエアの耐タンパー性には，機能の改変困難性と機密データの守秘性という２つの面がある．機能の改変困難性とはプログラムの機能に対する不当な改変が困難な性質であり，秘密データの守秘性とはプログラム内の秘密データを抽出することが困難な性質である．署名ソフトウエアの場合は，プログラム内部に現れる署名生成鍵に対する守秘性が重要である．尚，本研究では機密データの守秘性を扱うので，以後，耐タンパー性と言う場合は機密データの守秘性のことを言うことにする．

この守秘性を評価する方法として，プログラムの

実行過程で計算機内部に現れるデータ全てを署名生成鍵の候補としてチェックしていき，機密データを見つけるまでに要した時間や計算量で評価するという，全数探索を基にした方法が考えられている[1)2)]．この方法をランタイムデータ全数探索法と呼ぶことにする．

ランタイムデータ全数探索法による耐タンパー性評価を行うには，評価対象ソフトウエアの実行中に計算機の内部に現れるランタイムデータを抽出すること，抽出したランタイムデータ全てについて機密データであるかどうかをチェックすること，この二機能を持つツールが要求される．

ところで，Java は，モバイルコードがインターネットを介して流通することが一般化していくことも鑑みると，将来において最も重要なプログラミング言語の一つである．署名ソフトウエアを始め，機密データを扱う多くのソフトウエアが Java で開発され，クラスファイル等の形で配布されていくと思われる．Java クラスファイルは，Java 仮想機械の実装の上で動くため，Java のランタイムデータはこの Java 仮想機械中に現れるデータである．

本研究では，署名ソフトウェアの耐タンパー性評価法としてランタイムデータ全数探索法を採った上で，Java 仮想機械中に現れるランタイムデータを抽出するツールを開発した．そして，Java で記述した RSA 署名ソフトウエアに対して開発したツールを適用し，署名生成鍵をランタイムデータとして抽出する実験を行った．

以下，第 2 章ではツールの開発について，第 3 章では実験について述べ，第 4 章ではまとめと今後の課題について述べる．

2. ランタイムデータ全数探索による耐タンパー性評価ツール

ランタイムデータ全数探索による耐タンパー性評価ツールは図 1 に示すように，Java プログラムの実行過程中のメモリデータを捕捉する Data Extractor（これが Java 仮想機械ランタイムデータ抽出ツールである）と，捕捉されたデータが秘密データであるか否かを検証する Checker から成る．本稿では，主に Data Extractor の部分の実装について記述する．Checker におけるチェックの内容は評価対象であるソフトウエアに応じて定まる．評価対象が署名ソフトウエアである場合は，DataExtractor が抽出したランタイムデータ一つ一つについて，それが署名生成鍵であるかを，実際に署名を生成し検証していくことでチェックする．DataExtractor は評価対象であるソフトウエアの種類にかかわらず汎用的なものである．すなわち，署名生成プログラムに限らず，秘密データを扱う Java プログラム全般に対して，その秘密データ，あるいは秘密データの断片のダンプに使えるものである．尚，以下ではランタイムデータ抽出のことをダンプと呼ぶ．

2.1. DataExtractor

Data Extractor は，Java のデバッガ作成のためのアプリケーションインタフェースである JPDA(Java Platform Debugger Architecture)[3)]の実装を拡張してその上に構成したもので，JVM（Java Virtual Machine）メモリ上に出現したデータを捕捉して，型によって分類し，ダンプデータとして出力する機能を有する．Data Extractor 開発の技術的課題としては，既存の JDK(Java Developers Kit)に含まれる JPDA 実装では，JVM 中に現れるデータのうちで，各メソッドのローカル変数値しかダンプできない(ローカル変数値についてもダンプできない場合もある)点がある．そのため，JPDA を拡張して，オペランドスタック上の値，ヒープ上の整数型配列や BigInteger 等整数関連オブジェクトまでダンプ出来るようにする必要がある．

2.2. Checker

Checker は，ダンプデータを基に署名生成鍵の候補の集合を作り，各候補で署名の生成と検証を行うことで，署名生成鍵かをチェックする（署名生成鍵を見つけ出す）機能を有する．Checker にて署名生成鍵を検出するのに要した時間をデータ守秘性に関する耐タンパー性の定量的評価と位置づける．署名生成プログラムにおいて鍵を単一の配列やオブジェクトとして扱っている場合はダンプデータ中からそれらを見つけ出すのは容易であるが，そうでない場合は，鍵の断片がダンプデータ中に散在することになり，それらを組合せて鍵候補を作る必要がある．この際，組合せ的爆発による全数探索空間の巨大化を避けるために，組合せの方策を工夫する必要がある．本稿で述べるものは，プロトタイプとして開発した，特に組合わせ方策を考えない（つまり DataExtractor から出てきたデータをそのまま鍵候補として Check する）版である．

2.3. アルゴリズム

DataExtractor と Checker を使った，署名ソフトウエアに対するランタイムデータ全数探索による耐タンパー性評価のアルゴリズムの概略を示す

Ω: DataExtractor により抽出されたデータの集合

d_i: Ω 中のデータの組合せ

とする．

1) 評価対象の java プログラムを実行し，メモリ上に出現する全てのデータを捕捉し，ダンプする．すなわち，署名生成鍵とメッセージを署名生成ソフトウエアに与えて JVM 上で実行させ，ランタイムデータを DataExtractor により抽出する．

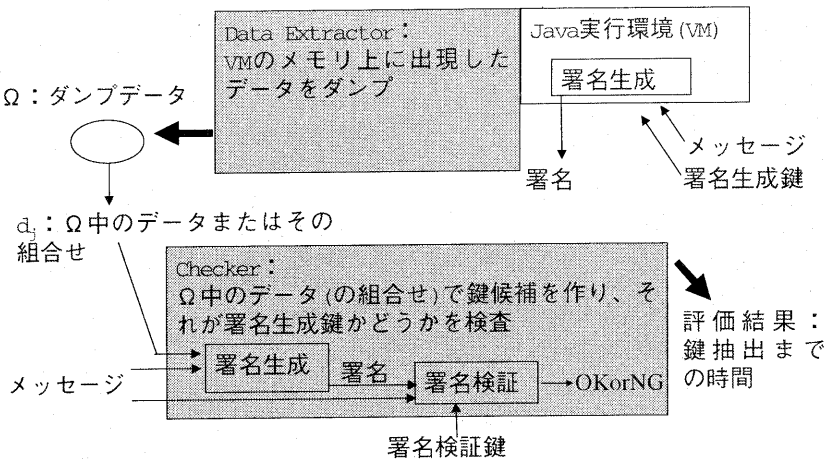

図1　ランタイムデータ全数探索による耐タンパー性評価ツールの構成

2)d_jとメッセージを署名生成関数に入力して，署名を生成する．

3)生成した署名と署名検証鍵を署名検証関数に入力して，署名が妥当であるか否かを検証する

4)署名が妥当である場合，d_jが署名生成鍵である

5)署名が妥当でない場合，次のデータd_{j+1}について，2)から繰り返す

6)署名生成鍵が見つかるまでの時間を耐タンパー性評価結果とする

2.4. Data Extractor　(Java 仮想機械ランタイムデータ抽出ツール)の実装

Data Extractor 開発の技術的課題は，JPDA を拡張して，オペランドスタック上の値，ヒープ上の整数型配列や BigInteger 等整数関連オブジェクトまでダンプ出来るようにする点であった．この課題を解決することを考慮し，DataExtractor（以下 DE と略記）の構成を図2のようにした．

DE は Target 側と Debugger 側二つに分かれている．ダンプ対象 Java プログラムと Target 側 DE が稼動する Target 側 VM，Debugger 側 DE が稼動する Debugger 側 VM，この二つの VM が並立して動いている状況で DE は稼動する．Java プログラム（正確にはクラスファイル）中の実行コードはバイトコードと呼ばれる．Debugger 側 DE は JPDA を使ってダンプ対象 Java プログラムのバイトコードのステップ実行を制御する．同時に共有ファイルまたは共有メモリを介して Target 側 DE にダンプ条件（バイトコードから割り出した，型やデータ出現位置に関する条件）を付けたダンプ要求を出す．Target 側 DE はダンプ対象 Java プログラムのステップ実行の間に VM の内側に直接アクセスし，データを捕捉して，共有ファイルまたは共有メモリを介して Debugger 側 DE に渡す．Debugger 側 DE はそれを整形しダンプデータとしてダンプファイルに出力していく．

Target 側 DE の構成を図3に示す．JVM の実装中には jdwp という C の関数があり，これがステップ実行の制御を司っている．Debugger 側 DE からステップ実行制御司令が来た時，jdwp は司令されたステップ実行の直後に dumper という C の関数を呼ぶように jdwp を改良した．dumper は新たに開発した関数で，Target 側 DE からデータ要求があった場合は，直前のステップ実行により生成されたデータを JVM メモリ上で捕捉し送出する．ShardDataManager という C の関数は共有メモリまたは共有ファイルを通じて Target 側 DE と Debugger 側 DE がデータをやり取り際のインタフェースを司る．

JVM のメモリ構成について簡単に述べる [5]．JVM の中では，スレッドが一つ起動する度に Java 仮想マシン・スタックと呼ばれるメモリ領域が確保される．そしてスレッドの実行においてメソッドが呼び出される度にフレームが Java 仮想マシン・スタック内に形成される．フレームにはオペランドスタックとローカル変数領域が取られる．この二つのメモリ領域上に現れるデータがランタイムデータである．バイトコードの実行はこの二つのメモリ領域上でデータを読み書きすることにより行われる．

3. 実験

開発した DataExtractor の動作を検証するための実験を行った．

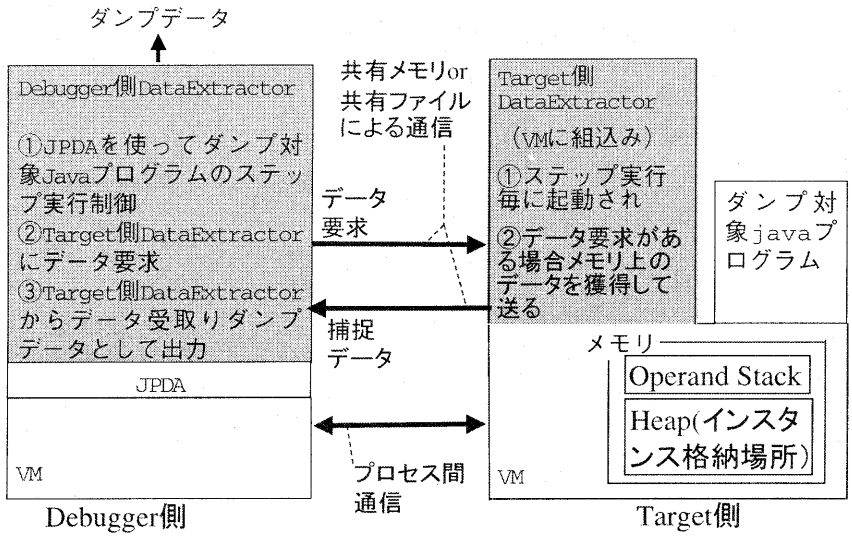

図2　DataExtractor の構成

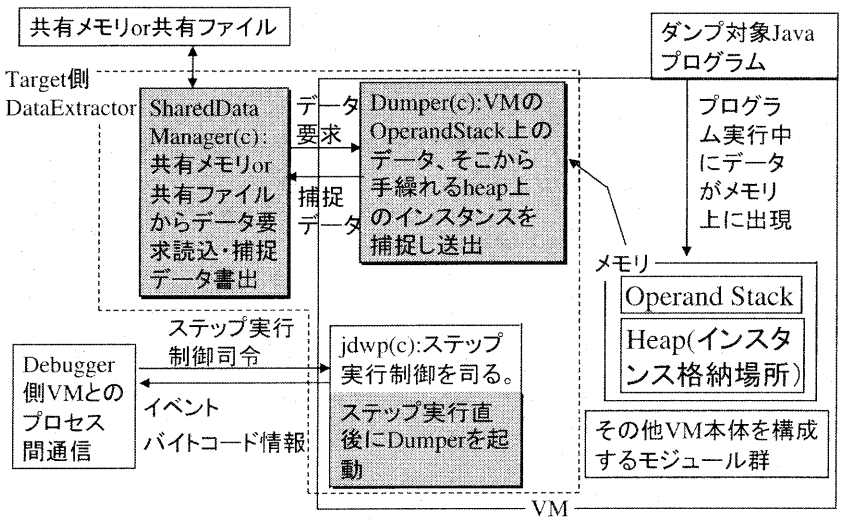

図3　Target 側 DataExtractor の構成

3.1. 実験対象プログラム

署名生成・検証のアルゴリズムとして最も広く使われているのは RSA 署名である．SHA-1/RSA が PKCS#1 の中で規定されており，日本の電子署名法に係る指針において記載された 4 つの署名アルゴリズムのうちの一つである．SunJCE やその互換版として米国以外で広く使われている Cryptix ライブラリ中には SHA-1/RSA の署名生成プログラムが実装されている．このプログラム中では，鍵を BigInteger オブジェクトとして持ち，鍵生成計算や検証計算は BigInteger クラスの演算メソッド(べき乗剰余算である modPow 等) を使って書かれている．ここでは，DataExtractor の動作検証実験の対象として，鍵を BigInteger オブジェクトとして持ち，鍵生成演算や検証演算を BigInteger クラスの演算メソッド(modPow 等)によって構成した Java プログラム（ハッシュ関数は含まない）を開発し，それに対して Data Extractor を適用した実験について記す．実験結果の

#ダンプ番号 : 型([は配列を示す) : ダンプ時に実行していたバイトコード : ダンプ時のメソッド : ダンプ値

```
62:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:0,0,223
78:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:0,0,19423
94:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:0,0,150495
130:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:0,41,-1342026785
146:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:0,5417,-1342026785
162:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:0,16586025,-1342026785
198:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:199,922555689,-1342026785
269:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:0,0,251
285:[I:iaload:java.math.BigInteger.stripLeadingZeroBytes(byte[])+95:0,0,15099

中略

616:[I:iaload:java.math.BigInteger.getInt(int)+33:38087,922555689,-1342026785
640:[I:iaload:java.math.BigInteger.bitLength(int[], int)+13:37091,-806412275,1005730555
657:[I:iaload:java.math.BigInteger.oddModPow(java.math.BigInteger, java.math.BigInteger)+77:7,25,81,241,673,1793,2147483647

中略

4751:[I:iaload:java.math.BigInteger.intArrayCmpToLen(int[], int[], int)+7:24286,-981014151,-1420014254,0,0,0
4756:[I:iaload:java.math.BigInteger.intArrayCmpToLen(int[], int[], int)+17:38087,922555689,-1342026785
4778:[I:iaload:java.math.BigInteger.oddModPow(java.math.BigInteger, java.math.BigInteger)+415:37091,-806412275,1005730555
4785:[I:iaload:java.math.BigInteger.oddModPow(java.math.BigInteger, java.math.BigInteger)+415:37091,-806412275,1005730555
4792:[I:iaload:java.math.BigInteger.oddModPow(java.math.BigInteger, java.math.BigInteger)+415:37091,-806412275,1005730555
4822:[I:iaload:java.math.BigInteger.oddModPow(java.math.BigInteger, java.math.BigInteger)+542:37091,-806412275,1005730555
4854:[I:iaload:java.math.BigInteger.squareToLen(int[], int, int[])+33:14906,1091654037,-578829730,0,0,0
```

署名生成鍵かどうか一つずつcheckすれば，下線部が署名生成鍵に対応する部分であることが直に判明

図4　署名生成 Java ソフトウエアに対する DataExtractor 適用実験

一つを図4に示す．

3.2. 検証実験結果の分析

図4の下線部のように署名生成鍵が抽出したランタイムデータの中に入っている．よって，ランタイムデータ中の int 配列型のデータについて，Checker プロトタイプを使って署名生成鍵であるかを一つずつ check していけば，署名生成鍵が判明する．DataExtractor の動作を検証するための実験であったが，その結果を分析することによって次のことがらがわかった．Cryptix 等 RSA 署名生成プログラムは，1)署名生成鍵を保持するオブジェクトから署名生成鍵を取り出して BigInteger としてセットし，2)メッセージも BigInteger としてセットし，3）BigInteger クラスの演算メソッドを使って署名生成計算を行う，という基本構造をしているものが多いと考えられるが，その場合，以下のような部分に耐タンパー性に関して脆弱な点があることがわかった．

- 署名生成鍵のセット段階でそれがランタイムデータとして Extract される
- modPow 等の BigInteger クラスの演算メソッド実行中に署名生成鍵がランタイムデータとして Extract される

4. まとめと今後の課題

署名ソフトウェアの耐タンパー性評価法としてランタイムデータ全数探索法を考えた上で，Java 仮想機械中に現れるランタイムデータを抽出するツールを開発した．そして，Java で記述した RSA 署名ソフトウエアに対して開発したツールを適用し，署名生成鍵をランタイムデータとして抽出できるという実験結果を得た．

- Cryptix 等，SHA-1/RSA の署名プログラムとして広く使われている Java ライブラリ中においては，一つの鍵データを一つの BigInteger オブジェクトとして扱っているものが多いと考えられるが，その場合，BigInteger が簡単にダンプされ，鍵データとして検出される．現に前章で述べたように，Cryptix と同様に鍵データを BigInteger オブジェクトで扱う RSA 署名生成プログラムを作成し，今回開発した DataExtracter と checker プロトタイプにかけたところ，署名生成鍵である BigInteger オブジェクトのダンプと検出が簡単に成された．
- 鍵データを BigInteger オブジェクトではなく int 配列や long 配列といったオブジェクトとして持っていても，一つのオブジェクト内に鍵データとしてまとまって存在する限り簡単にダンプできて，検出できる．やはり今回開発した DataExtractor と checker プロトタイプによって簡単にダンプされ，検出された．

署名生成プログラムの耐タンパー性を高めるためには，一つの鍵データを幾つかのデータに分割して持ち，鍵データが断片ごとにダンプはされても，一つのまとまったデータとしてダンプされないようにすることが耐タンパー化基本方策になる[1]．しかしこのような鍵分割の方策は，Checker における組合せの方策と「いたちごっこ」の関係にあるとも言える．したがって，どのような鍵分割が耐タンパー化にとって有効であるか，あるいは逆に，Checker がデータを組合せて鍵の候補を出す方策としてはどのようなものが強力か，

といった事柄について研究をすることは意義がある．

なお，本稿で報告した DataExtractor の開発は，IPA（情報処理振興事業協会）の情報技術開発支援事業（平成 13 年度）の支援を受けて行われた．

文　献

[1] 赤井健一郎，松本勉，"耐タンパーソフトウェアの全数探索型強度評価法，" 信学技報，Vol.99, No.699, IT99-81, Mar.2000.

[2] 赤井健一郎，松本勉，"共通鍵暗号ソフトウェアの全数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 2000（CSS2000）論文集，pp.205-210, Oct.2000.

[3] JavaTM Platform Debugger Architecture, http://java.sun.com/products/jpda/.

[4] Cryptix3, http://www.cryptix.org/products/cryptix31/.

[5] T. Lindholm and F. Yellin, "The Java(TM) Virtual Machine Specification (2nd Edition)," Addison Wesley, May 1999.