社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－DPS－130 (26)
2007－CSEC－ 36 (26)
2007／3／1

Webアプリケーションにおける言語レベルの動的情報フロー制御

吉濱佐知子 †‡　吉澤武朗 †　渡邊裕治 †　工藤道治 †　小柳和子 ‡

† 日本アイ・ビー・エム東京基礎研究所
242-8502 神奈川県大和市下鶴間 1623-14
{sachikoy, kudo}@jp.ibm.com

‡ 情報セキュリティ大学院大学
221-0835 神奈川県横浜市神奈川区鶴屋町 2-14-1
{mgs054503, oyanagi}@iisec.ac.jp

あらまし 3層 Web アプリケーションにおいて，アクセス制御によってバックエンドのデータベース内に記録されている機密情報を保護していても，一旦データが取得された後にそのデータが適切に扱われるかどうかは，アプリケーションの実装に依存する．本論文では，Java^{TM} 言語においてバイトコードを書き換えにより情報フロー制御を行なう機能を挿入することにより，実行時に Web アプリケーションの実行による不適切な情報の伝播を防止する方式を提案する．動的な情報フロー制御機構とデータベースアクセスを連携することにより，データベースから読み出された情報を，アプリケーションの実行を通して，一定の情報フロー制御ポリシーの下に保護することが可能になる．

Language-Based Dynamic Information Flow Control for Web Applications

Sachiko Yoshihama †‡　Takeo Yoshizawa　Yuji Watanabe
Michiharu Kudoh †　Kazuko Oyanagi ‡

†IBM Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242-8502 Japan
{sachikoy, kudo}@jp.ibm.com
‡Institute of Information Security
2-14-1 Tsuruyacho, Kanagawa-ku, Yokohama-shi, Kanagawa 221-0835 Japan
{mgs054503, oyanagi}@iisec.ac.jp

Abstract In the three-tier web application model, sensitive information in the back-end database is intended to be protected by the access control mechanisms at the database management system. However, the access control is often not effective since once the sensitive information is retrieved, protection of the information depends on the behavior of the web application. This paper proposes a dynamic information flow control mechanism for Java-based web applications, especially those involve database accesses. The approach inserts inline reference monitors (IRMs) into the program by modifying Java bytecode, and then tracks and controls information flow within the program at runtime. Bytecode rewriting technique allows support for legacy software without source code, and being independent of Java virtual machine implementations. The proposal enables protection of sensitive information throughout execution of applications.

1 はじめに

一般的な 3 層 Web アプリケーションでは，ユーザの個人情報やクレジットカード番号といった機密情報はデータベースに格納されており，データベースへのアクセス制御により保護されることが期待される．しかし Web アプリケーションがデータベースからデータを取得した後は，その情報をどう扱う

かということはアプリケーションの挙動にかかっている．不注意な開発者によって書かれたアプリケーションは，読み出した機密情報を不適切な場所に出力してしまうかもしれない．

また，現在の多くのWebアプリケーションは，単一のアカウントを使ってデータベースにアクセスするように設計されているという問題がある．特に，データベースに対してコネクションを開く処理は時間が掛かるので，パフォーマンスの最適化のためにコネクションプーリングをして複数のトランザクションの間でコネクションを使いまわすようにすることが多く行なわれている．コネクションプーリングを行なう場合，Webアプリケーションの相手にしているユーザのアイデンティティに関わらず，単一のアカウントによってデータベースにアクセスするため，データベースにおけるアクセス制御は有効ではない．

図1は，情報漏えいを起こすバグのあるプログラムの例である．このプログラム例は，非常にシンプルなWebサーバ上のオンラインショップで，ユーザ名，購入するアイテム名をHTTPリクエストから読み出し，データベースからユーザに対応する（事前に登録された）クレジットカード番号を読み出して，購入処理を行なう．processPurchaseメソッドでは，クレジットカード番号の有効性をチェックし，問題のあるとき（例えばクレジットカード番号の有効期限が切れているとき）はfalseを返し，問題の無いときはtrueを返して処理を続行する．ここで，HTTPリクエストから受信したユーザの情報は非機密，クレジットカード番号は機密情報とする．(ただし，このメソッドの実行前に適切なユーザ認証が行なわれ，クライアントとサーバ間のコミュニケーションチャネルはSSLやTLSで保護されているため，HTTPリクエストの内容は信頼できるもの仮定する．)

一見正しいプログラムであるが，不用意にクレジットカード番号をユーザに送信したりログに出力している点は機密性という観点から望ましくない．

本論文で提案するシステムは，Java^{TM} プログラム上でこのような望ましくない情報フローを防ぐためのものである．上記の例で言えば，システムはクレジットカード番号がHTTPレスポンスやログファイルに出力される前に，望ましくない情報フローの発生を検出して処理を中断する．また，特に実用性に配慮し，1) ソースコードのない既存のJavaアプリケーションを改造せずに適用可能とし，2) 実行環境（Java VM）に依存せず，また3) ポリシーに基づいた情報フロー制御と適切な機密解除 [11] を可能とする．

```
protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ... {
  String user = request.getParameter("user");
  String item = request.getParameter("item");
  PrintWriter pw
   = new PrintWriter(response.getOutputStream());
  ...
  String credit = getCreditCardInfoFromDB(user);
  boolean b = processPurchase(user, item, credit);

  if(b){    // succeeds
    pw.println("Purchase Succeeded: <br/>");
    pw.println("Name: " + user + "<br/>");
    pw.println("Item: " + item + "<br/>");
    pw.println("Credit Card: " + credit ); // bad
  }else{    // failed
    printlog("Invalid credit card: " + credit); // bad
  }
  ...
}
```

図 1: Webアプリケーションプログラム例

以降の章は次のように構成されている。2章では、提案方式の概要について、3章では、バイトコード命令ごとの情報フロー追跡の詳細について，4章では，JDBCモニタによる動的なラベル付与について，5章ではプロトタイプ実装について、6章では、先行研究について考察する。7章ではまとめと今後の課題について述べる。

2 Javaにおける情報フロー制御機能

本提案では，IFC-J [13] を拡張し，実行時に動的に情報フローの制御を行なう．以下にその概要を述べる．

Javaのアプリケーションが実行される場合，まずJava仮想マシン（JVM）が起動され，初期化コードからアプリケーションのエントリポイントとなるメソッドが呼び出される．Javaスタンドアロンアプリケーションの場合，エントリポイントは実行されるクラスのmain() メソッドであり，Webアプリケーションの場合はServletインタフェースで定義されるdoGet, doPostメソッドなどとなる．プログラムの実行が進むにつれ，メソッドは別のメソッド，またはライブラリなどを呼び出し，各メソッドの実行は，正常復帰または例外の発生によって完了する．最終的に，補足されない例外が発生するか，main() メソッドから復帰することによって，アプリケーショ

ンの実行が完了する．

プログラム外部への出力または入力は，APIを介して行なわれる．ある外部環境S（ファイル，ネットワーク上の他のノード，データベースなど）から入力された情報が別の外部環境Dに出力されるとき，プログラムの実行を介してSからDへの情報フローが発生していると言うことができる．本提案の目的は，このような情報フローを制御することにある．

IFC-JはJavaプログラム実行時の望ましくない情報フローを防止するために、1) プログラムに対する入力元・出力先となる外部環境SとDを定義し、それらのセキュリティラベルを定義し、2) ラベル間の情報フローポリシーを強制し、3) プログラム実行中の情報フロー伝播を追跡する仕組み、の3つを提供する。

2.1 入出力のラベル付けポリシー

Javaプログラムへの基本的入出力には、コマンドライン引数、標準入力、標準出力、ファイルおよびネットワークへのアクセスなどがある。またこれらの入出力を包含したロギング、データベースアクセス、Servlet APIでの入出力、など上位のAPIも存在する。

いずれの場合もプログラムにとっての入出力は、Java APIのメソッド呼び出しなど、ライブラリとのインタラクションによって発生すると考えられる。よって、ここでは、Java APIを介する入出力先を"java:クラス名表現" といういう擬似URIで表現する。また、ファイルやネットワークへのアクセスは同じAPIであっても対象ファイル／ホスト・ポートによって意味合いが異なってくるため、アクセス対象をURLで表現する。

2.2 情報フローポリシー

提案方式はポリシー非依存であるため、情報フローポリシーとしては、要件にしたがってBell-LaPadula [9]やBiba [10]などの異なるポリシーを採用することが可能である。以下の例では簡便のため、2つのラベルHIGH、LOW間の秘匿性を考慮したシンプルなBell-LaPadulaモデルに基づくポリシーを採用する。つまり、同ラベル間とLOWからHIGHへの情報フローは許されるが、HIGHからLOWへの全ての情報フローは許されない。なお明示的にラベルの付いていない変数やオブジェクトにはNONEという仮想的なラベルが付いているものとして扱い、HIGHまたはLOWからの情報フローが生じた時点でそのラベルが伝播される。

2.3 情報フロー追跡の仕組み

情報フロー制御機構のアーキテクチャを図2に示す。本方式では，Javaのバイトコードを書き換えることにより，情報フロー追跡を行なうための特殊なinline reference monitor (IRM) コードをアプリケーション中に埋め込む．プログラムの実行時には、挿入されたIRMコードがアクセス制御モジュール(ACM) の呼び出しを行い、情報フローの追跡とポリシーの強制を行なう。

図2の下部は、JVMの持つ内部構造を表している。JVMは実行中の各スレッドごとの実行中の状態を保持する構造を持ち、それぞれのスレッドtの実行Frameを保持するJVM Stack (js^t) とプログラムカウンタ (pc^t) を持つ。メソッド呼び出しが行なわれる時に新しいFrame fr_i^tが作成され、js^t上にpushされる。それぞれのfr_i^tはローカル変数テーブルlv_i^tとオペランドスタックos_i^tを持つ。また、オブジェクトの実体はヒープ領域に格納される．

ACMはJVMの持つ内部構造に対応するデータ構造 ($l(pc^t), l(fr_i^t), l(os_i^t), l(lv_i^t)$) を持ち、実際にプログラム中で操作される値の代わりに、値に関連付けられたセキュリティラベルを保持する。また、オブジェクトやそのフィールドに関連付けられたラベルを管理するためのObject Label Table (OLT)、配列要素のラベルためのArray Label Table (ALT)、そして各クラスのスタティックフィールドのためのClass Label Table (CLT) を持つ。

実行時に，プログラムに埋め込まれたIRMコードがACMの機能を呼び出し，ACMとJVMの状態を同期させて，ACMがJVMで持つデータのラベルを管理するようにする．この機構により，JVMに手を加えることなく，JVM上で演算に使用されているデータのセキュリティラベルを管理することができる．

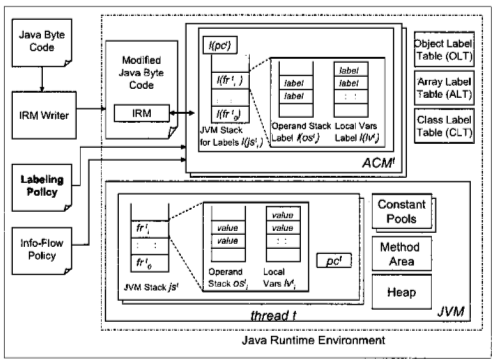

図 2: アーキテクチャ

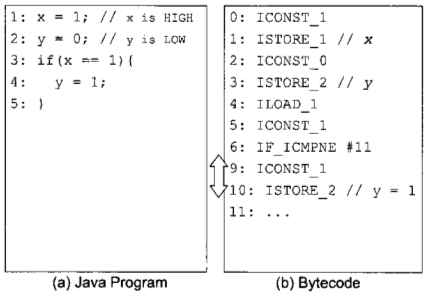

図 3: Method Invocation

3 バイトコード命令ごとの情報フロー追跡

JVM には約 200 のバイトコード命令があるが，データ型毎に同じ命令が複数定義されるなど意味的には重複がある．以下に，代表的なバイトコード命令とその追跡を行なう ACM の処理について説明する．

3.1 演算とローカル変数操作

メソッド内での演算やローカル変数の操作は、ACM 内に保持したセキュリティラベルをオペランドスタックやローカル変数の操作に合わせて伝播することで情報フローをトラックする。すなわち，LOAD 命令，STORE 命令では，データが os_i^t と lv_i^t の間で伝播されるのに合わせて $l(os_i^t)$ と $l(lv_i^t)$ の間でラベルを伝播する．2 項演算が行なわれる場合，2 つの値が os_i^t 上にロードされ，演算結果が os_i^t 上に置かれる処理にあわせ，ACM では 2 つの値のラベルを $l(os_i^t)$ にロードし，この 2 つのラベルを合成したラベルを演算結果のラベルとして $l(os_i^t)$ に置く．

3.2 オブジェクト/フィールド参照

JVM では PUTFIELD 命令，GETFIELD 命令によって，オブジェクト・フィールドとオペランドスタック間の情報が伝播されるので，ACM では，$l(os_i^t)$ と OLT の間でラベルを伝播する．同様に、スタティックフィールドや配列要素の読み書きは，対応する命令において $l(os_i^t)$ と CLT, ALT の間でラベルを伝播することで管理する．

3.3 メソッド実行

メソッド実行に伴う情報フローは，メソッド呼び出しの引数と，その復帰値，もしくは例外を介して発生する．メソッド foo() が baa() を呼び出す場合，foo() の視点からは，メソッドの実引数が出力であり，復帰値が入力となる．逆に baa() の視点からは，メソッドの仮引数が入力であり，復帰値が出力となる．そこで，入出力ラベル付けポリシーはこの 4 つを区別したラベル付けを可能にする．メソッドに対する入力が行なわれる場合，明示的なポリシーが指定されていれば，入力値にそのラベルが関連付けられる．逆に，メソッドからの出力が行なわれる場合，出力値と出力先のラベルを比較し，情報フローポリシー違反を検出した場合には，情報フロー例外を発生して処理を中断する．たとえば，ラベル HIGH のデータを，ラベル LOW に関連付けられているメソッドの実引数に指定した場合に，情報フロー違反が発生する．

3.4 分岐・ループでの暗黙的フロー

プログラム中で明示的に代入が行なわれなくても，プログラムの制御構造を通じて間接的に情報フローが生じることを，暗黙的フロー (implicit flow) と呼ぶ [1][14].

図 3(a) の例では，プログラム実行後に y の値から x が推測可能である．このコードをコンパイルしたものが図 3(b) である．6 行目の IF_ICMPNE によって分岐が行われるため，分岐先までの 9,10 行目が x の値の影響を受ける範囲である．ここで，実行中の命令を指す pc^t のラベルをデータに伝播することで，暗黙的フローを追跡することが可能になる．

4 データベースとの連携

JDBC (Java DataBase Connectivity) は，共通化された API によってデータベースやドライバごとの差異を吸収することにより，特定のデータベースに依存しないアプリケーションの実装を可能にする仕様である．

APM4JDBC (Application Privacy Monitoring for JDBC) [15] は，JDBC API と JDBC Driver の間をフックして，カスタマイズされた処理を追加する汎用的なモニタ・アーキテクチャである．モニタの機能としては，データベースへのアクセス制御，データベースアクセスのログ記録，などが考えられる．また，アプリケーションにアクセスするユーザの情報などを ContextHandler に通知することで，Web ユーザのアイデンティティに基づくアクセス制御などが可能となる．

本提案では，APM4JDBC を利用して，データベースのクエリ結果に対してセキュリティラベルを付与するように拡張した（図 4）．ここで，Access Monitor にプラグインされる Access Controller において，データベースのクエリ結果である情報に，ラベル付けポリシーに基づいてセキュリティラベルを関連付ける．この機構により，データベースのカラムごとに異なったラベルを付けたり（例：クレジットカード番号のカラムのみを HIGH にする），ユーザ毎に読み出した情報に別のラベルを付与することによって，あるユーザに属する機密データが別のユーザに出力されないようにする，などの制御が可能である．

5 プロトタイプ

提案システムのプロトタイプを，Apache Tomcat 上に実装し，Web アプリケーションクラスローダを改造することによって動的にバイトコードを書き換えるようにした．ACM は Java のクラスとして実装した．ここで 1 章の例題アプリケーションを実行し，情報フロー違反を検出することを確認した．次にクレジットカード番号をマスクする mask() メソッドを導入し，このメソッドの復帰値に機密解除ポリシーを定義することで，正しくプログラムを実行できることを確認した．

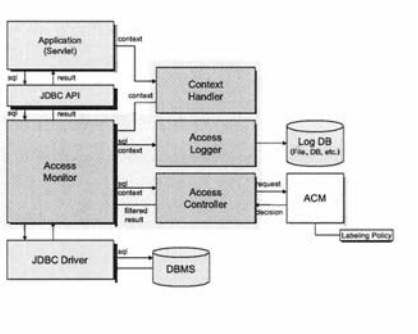

図 4: APM4JDBC

6 関連研究

情報フロー分析・制御は、静的に行なう手法と、実行時に動的に行なう手法に大別される。静的手法はプログラムの実行前に行なうため、実行時のオーバーヘッドがないという利点があるが，実行時の動的な状態を含む情報フローを完全に解析するのは難しい．文献 [8] や [16] では Java バイトコード命令（またはそのサブセット）の上での情報フロー分析を行なっている．[7] はセキュリティ型付言語をコンパイルした後にもその性質を保持することを証明している．Jif[2] では Java 言語を拡張しセキュリティラベルを記述することによって、情報フロー制御を可能にしている。

動的手法は、実行されたパスのみが検証されるため、全ての実行パスについての網羅的な分析を行なうことが難しく，実行時オーバーヘッドが発生するという欠点がある。一方で、実行時のプログラムの状態をすべて利用できるため、より正確な判断が可能である。文献 [1] はオペレーティングシステムを改造し、各機械語命令の実行ごとに情報フローをトラックすることによって、プログラム内での情報フ

ローをトラックする仕組みを提案している。同様の方式を Java に適用することも可能だが、その場合は JVM 自体を改造する必要があるため実行環境の実装への依存が発生する。

Erlingsson ら [3] は IRM によって Java2 セキュリティ[4] と同等のアクセス制御を行なっている. Hadler ら [5] は IRM を利用し、Java プログラムに対する外部入力値に taint フラグやセキュリティラベルを付与することによる情報フロー制御を提案している。本論文の提案は Hadler らの方式に基づくが、最大の違いは情報フロー制御を行なう粒度であり、Hadler の方式では Java のオブジェクト単位にラベルが付与されるのに対して、本論文の方式では整数や浮動小数点数を含むプリミティブな値や配列要素のそれぞれといった細かい単位での情報フロー制御が可能となっている。

7 まとめと今後の課題

本論文で提案した方式では，複数のメソッド呼び出しに渡って，オブジェクトだけでなく整数や浮動小数点を含む primitive 値や配列などの情報フローを細かい粒度で制御できるという利点がある．一方で，本方式には幾つかの課題がある．まず，バイトコード命令単位での IRM では，情報フローを完全に把握するために JVM 上で実行される各命令ごとに対応する情報フロー制御処理を行なう必要があるため，オーバーヘッドが大きい．オーバーヘッドを削減するためには，ACM による実行時チェック処理を最適化し，情報フロー制御処理を呼び出す頻度を低減するなどの工夫が必要になると思われる．ただし，実行速度の低下はテスト環境で使用する場合には大きな問題ではないが，テストの網羅性が重要になる．

また，動的手法のみでは暗黙的フローを完全に排除できない問題があり，静的手法を組み合わせて使うことも考えられる．

次に，現在の入出力ラベル付けポリシー指定による機密解除 (declassification) の仕組みは，プログラムの改造は必要ないが，ポリシー定義者にプログラムの構造と各メソッドの機能についての知識を要求する．また，人的誤りによる情報フロー違反の可能性を排除できない．ソースコードに対する知識のない状態で効率よく，かつ安全に機密解除を行なう方式は今後の課題である．

謝辞

本研究の内容に関して一緒にご議論いただいた日本 IBM 東京基礎研究所および情報セキュリティ大学院大学の方々に感謝します。また、本研究は、経済産業省、新世代情報セキュリティ研究開発事業の研究として行われたものです。

参考文献

[1] Y. Beres, C.I. Dalton, Dynamic Label Binding at Run-time, NSPW 2003.

[2] L. Zheng et al., Using Replication and Partitioning to Build Secure Distributed Systems, IEEE Sympo. on S&P, 2003.

[3] U. Erlingsson, F.B. Schneider, IRM Enforcement of Java Stack Inspection, IEEE Sympo. on S&P, 2000.

[4] L. Gong, et al. Going Beyond the Sandbox: An Overview of the New Security, Usenix Sympo. 1997.

[5] V.Haldar, D. Chandra, M. Franz, Dynamic Taint Propagation for Java, Annual Computer Security Applications Conference 2005.

[6] A. Sabelfeld, A.C. Myers, Language-Based Information Flow Security, IEEE J. on Selected Areas in Comm, VOl.21, No.1, Jan. 2003

[7] G. Barthe et al, Deriving an Information Flow Checker and Certifying Compiler for Java, IEEE Sympo. on S&P, 2006.

[8] 小林直樹 白根慶太, 低レベル言語のための情報流解析の型システム, コンピュータソフトウェア Vol.20, No.2 (2003), pp.2-21

[9] D.E. Bell, L.J. LaPadula, Secure Computer System: Unified Exposition and Multics Interpretation, MITRE MTR-2997, 1976.

[10] K.J. Biba, Integrity Considerations for Secure Computer Systems, MTR-3153, June 1975.

[11] P.Li, S. Zdancewic, Downgrading policies and relaxed noninterference, POPL 2005.

[12] W.Landi, B.G. Ryder, A safe approximate algorithm for interprocedural pointer aliasing, PLDI 1992.

[13] S. Yoshihama, M. Kudoh, K. Oyanagi, Inforation Flow Control for Java with Inline Reference Monitors, Computer Security Symposium 2006 (CSS 2006).

[14] A.C. Myers, JFlow: Practical Mostly-Static Information Flow Control, POPL'99.

[15] Application Privacy Monitoring for JDBC (APM4JDBC), http://www.alphaworks.ibm.com/tech/apm4jdbc

[16] S. Genaim and F. Spoto, Information Flow Analysis for Java Bytecode, VMCAI2005.