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

2007－ARC－172 (39)
2007－HPC－109 (39)
2007／3／2

動的なインフォメーションフロー制御による情報漏洩防止手法

栗田弘之† 塩谷亮太† 入江英嗣††
五島正裕† 坂井修一†

従来の情報漏洩防止手法はプログラマの悪意やプログラムの脆弱性の有無，耐タンパ性といったプログラムの信頼性に依拠しているが，一般にプログラムの信頼性を保証することは難しい．本稿では，プログラム実行時のインフォメーションフローを動的に追跡・制御することによる情報漏洩防止手法を提案する．インフォメーションフローの追跡・制御はプロセッサと OS が行うため，提案手法はプログラムの信頼性に依拠しない．また，システムコールの形でプログラマへのインタフェースを提供することにより，プログラムの機能が過剰に制限されることを防止する．これにより，ユーザの利便性と情報漏洩の防止を両立する．

Dynamic Information Flow Control for Preventing Information Leakage

Hiroyuki Kurita,† Ryota Shioya,† Hidetsugu Irie,†† Masahiro Goshima† and Shuichi Sakai †

Conventional techniques to prevent information leakage are based on the reliability of programs. However, it is difficult to guarantee the reliability of a program since it depends on mane aspects such as program's vulnerabilities, tamper resistance, programmer's malice. In this paper, we focus on the information flow of programs and propose a novel technique to prevent information leakage. The technique dose not depend on the reliability of programs because the technique tracks and controls the information flows dynamically. In addition, the technique provides the interface to programmers and prevents excessive restriction to various functions of programs.

1. はじめに

コンピュータシステムが扱う情報は多様化しており，個人情報や企業の機密情報，商用マルチメディアコンテンツなどにおいては，それらの漏洩が社会問題となっている．

コンピュータシステムから情報が漏洩する原因は，ユーザの悪意や過失をはじめとして，スパイウェアやトロイの木馬型プログラムといった悪意あるプログラム，脆弱性を持ったプログラムに対する外部からの攻撃など，多岐にわたる．従来の情報漏洩対策では，個々の原因に応じていくつかの手法が用いられており，主な手法として，OS によるアクセス制御と Windows Media DRM のようなディジタル著作権管理システムがあげられる．

OS によるアクセス制御では，ユーザやプログラム毎にファイルなど各種リソースへのアクセスを制御し，不正なアクセスを防止する．しかし一方で，あるリソースへのアクセスを許可されたプログラムが，そのリソースから読み込んだデータをどう利用し，どこに出力するかに関しては制御できない．そのため，プログラムがデータを漏洩するか否かは，プログラマの悪意やプログラムの脆弱性の有無に依拠し，一般にこれらの存在を否定することは困難である．

ディジタル著作権管理システムでは，音楽データなどのコンテンツが暗号化された状態で配布され，専用プログラムがそれを復号化することで，コンテンツを再生する．一般にディジタル著作権管理システムでは，コンテンツを復号可能なプログラムを専用プログラムに限定し，その専用プログラムの信頼性を高めることで，コンテンツを保護している[2]．しかし，復号化処理をプログラム内部で行っているため，その強度はプログラムの耐タンパ性に依拠する．プログラムの耐タンパ性は暗号強度に比べ，その向上が困難であり，リバースエンジニアリング等の手段によってコンテンツが漏洩する恐れがある．また，コンテンツを復号可能なプログラムが専用プログラムに限定されることは，ユーザの利便性を損なう要因となる．

本稿では，プログラム実行時の情報の流れ (インフォメーションフロー) に着目した情報漏洩防止手法を提案する．提案手法では，プログラム実行時にプロセッサと OS がインフォメーションフローを追跡，制御することにより，保護するべきデータがプログラム外部に不用意に出力されることを防止する．提案手法の特徴を以下に示す．

- 情報漏洩防止機能をプロセッサと OS によって提供

† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†† 科学技術振興機構
Japan Science and Technology Agency

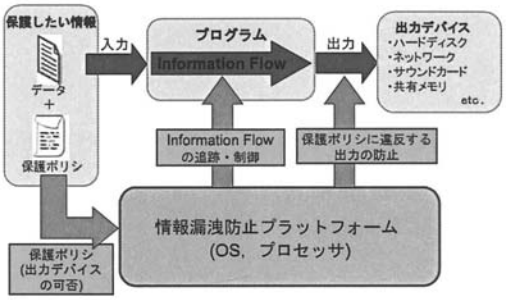

図 1 情報漏洩防止プラットフォーム概念図

し，その強度はプログラマの悪意やプログラムの脆弱性の有無，耐タンパ性といったプログラムの信頼性に依拠しない．

- 保護するべきデータが出力される際には，著作権者などが定める保護ポリシに従ってその可否を判断するため，ユーザの悪意や過失に依拠しない．
- システムコールの形でプログラマへのインタフェースを提供し，プログラム機能の過剰な制限を防ぐ．

以下，まず第 2 章で本稿がとるアプローチについて述べる．第 3 章では，インフォメーションフローについて説明し，第 4 章で提案手法の詳細について述べる．第 5 章で関連研究について説明した後，最後に第 6 章でまとめと今後の課題について述べる．

2. 情報漏洩防止に向けたアプローチ

2.1 情報漏洩防止プラットフォーム

情報漏洩の防止に向けて本稿がとるアプローチの概念図を図 1 に示す．

プログラムは，入力されたデータに対して定められた処理を行い，プログラム外部に何らかの出力を行う．たとえば音楽再生プログラムであれば，エンコードされた音楽データを入力として読み込み，デコードなどの処理を行った後，システムコールを介してサウンドデバイスのデバイスドライバに出力する．

ここで，プロセッサや OS から構成されるプログラム実行環境が，プログラム実行時の情報の流れ (以下，インフォメーションフロー) を動的に追跡ないし制御できるならば，プログラムの出力に注目して情報漏洩を防止することが可能である．それは，インフォメーションフローの追跡によって，プログラムが出力しようとするデータと，プログラムに入力されたデータとの依存関係がわかり，プログラム実行環境が出力の正当性を検証できるためである．本研究ではこのようなプログラム実行環境を情報漏洩防止プラットフォームと呼び，その実現を目的とする．

このプラットフォームを用いると，先に述べた音楽再生プログラムの例であれば，プログラムが音楽データを不正にネットワークやディスクドライブに出力しようとした場合に，それを検出，防止することが可能となる．

情報漏洩防止プラットフォームの特徴を以下に示す．

- アクセス制限やディジタル著作権管理などの既存技術とは異なり，プログラムの信頼性を前提としない．
- 一般のアプリケーションプログラムだけでなく，スパイウェアやトロイの木馬のようなウイルスに対しても有効である．
- ファイルからの入力だけでなく，ユーザのキー入力やネットワークからのストリーミングデータなど，プログラムに入力されるあらゆるデータが保護の対象となる．

なお，プログラムからの情報伝達は必ずしもデータ出力によってのみ起こるわけではなく，カバート (隠れ) チャネルと呼ばれる情報伝達経路が存在する．カバートチャネルの例としては，プログラムの実行時間やメモリ・ディスクの消費量，電力消費量などがあるが，本稿ではこれらカバートチャネルによる情報漏洩は範囲外とする．

2.2 プラットフォームの実現に向けた要件

本稿では，情報漏洩防止プラットフォームの実現に向けた要件のうち，基本的にインフォメーションフローの追跡・制御に議論を限定するが，ここでは，それ以外の要件について簡単に述べる．

2.2.1 保護ポリシ

情報漏洩防止プラットフォームでは，あるデータを保護したい場合，そのデータに関して，どのデバイスへの出力を認め，どのデバイスへの出力を禁止するかを決めておく必要がある．こうした出力先デバイスの可否情報を，本稿では保護ポリシと呼ぶ．

保護ポリシの例としては，「ネットワークへの出力の禁止」や，「認証されたサウンドデバイスのデバイスドライバへの出力のみ許可」，「ディスクへの保存は暗号化を行う場合のみ許可」などが考えられる．保護ポリシはデータの所有者や著作権者が記述し，データと保護ポリシは常に対応付けて保存されることを想定している．

2.2.2 OS およびデバイスドライバの役割とその完全性

プログラムの出力の可否は，保護ポリシを基に OS が判断する．また詳細は後述するが，インフォメーションフローの追跡・制御においても OS は重要な役割を果たす．そのため，OS が悪意を持って改竄された場合，情報漏洩の防止を保証できない．本研究では，TPM[6] などのハードウェアを起点としたセキュリティ技術によって OS の完全性が保証されることを仮定する．

また，プログラムからの出力は最終的に出力デバイスのデバイスドライバに委ねられる．そのためデバイスドライバについてもその正当性と完全性が保証される必要があり，電子署名等の手段によってこれらが満たされるものとする．

3. インフォメーションフロー

インフォメーションフローはプログラム実行時の情報

```
if(x == true)
    y = true;
else
    y = false;
```

図 2　x → y の暗黙的フローが発生するコード

の流れを意味し，データフローに付随する明示的 (explicit) フローと，コントロールフローに由来する暗黙的 (implicit) フローに分類される．本章では，これら二つのフローついて説明する．

3.1 明示的フロー

プログラムの実行においては，数値演算や論理演算など多くの演算が行われる．たとえば，$y = x + 5$ という演算について考える．このとき，演算後の y の値から x を決定できるため，この演算によって x の情報が y に伝わっていると考えることができる．このようなデータフローに付随する形の情報の流れを明示的インフォメーションフロー (以下，明示的フロー) と呼ぶ．

明示的フローによって伝わる情報には，不完全な情報も含まれる．たとえば，$y = x \% 2$ という演算では，演算後の y から x を完全に決定することはできない．しかし，x に関する情報の一部が y に伝わっており，このような場合にも x から y への明示的フローが存在する．

3.2 暗黙的フロー

図 2 に示したコードの実行を考える．x がブール変数であれば，コード実行後，y の値は x と一致する．そのため，明らかに x から y へのインフォメーションフローが存在する．しかし，前節で述べた明示的フローとは異なり，x と y の間に明示的な代入・演算関係はない．

この例のように，条件分岐を基点とし，分岐後のパスで行う処理が異なることによって発生するインフォメーションフローを暗黙的インフォメーションフロー (以下，暗黙的フロー) と呼ぶ．暗黙的フローは，条件分岐において条件として使われた値 (先の例では x) とその条件分岐に依存した処理のデスティネーション (先の例では y) の間に存在する．そのため，コントロールフローに由来するインフォメーションフローであるといえる．

3.3 暗黙的フロー追跡の困難性

図 2 のコードと同様に，図 3 に示したコードにも x から y への暗黙的フローが存在する．ここでこの暗黙的フローを，条件分岐後に行う処理に注目することによって，動的に追跡することを考える．

図 3 のコードを実行したときの挙動は以下のようになる．

- x が true のとき，条件分岐後，y への代入を行う．
- x が false のとき，条件分岐後，いかなる処理も行わない．

このため，条件分岐後の処理に注目しても，x が false のときには，x から y への暗黙的フローを認識できない．

暗黙的フローの発生は分岐後の実行パスにおける処理の違いに起因するため，「処理を行わないパス」を通った場合にも暗黙的フローは発生する．一般にすべての

```
y = false;
if(x == true)
    y = true;
```

図 3　x → y の暗黙的フローが発生するコード (2)

暗黙的フローを認識するためには，条件分岐後にとり得るすべてのパスに関して，そのパスを通ったときに変更される状態 (変数の値など) を解析する必要がある[7]．そのため，プログラムが実行時に通過するただ一つのパスを観測するだけでは，すべての暗黙的フローを認識することはできない．

次章では，暗黙的フローを追跡するのではなく，暗黙的フローの発生期間に注目した制御手法について述べる．

4. 提案手法

前章でのインフォメーションフローに関する考察をふまえ，本章では，以下のようなインフォメーションフロー追跡・制御手法を提案する．

- 保護するべき情報を識別するためのタグ (出力制限タグ) を命令実行時に伝搬させることで，明示的フローを追跡する．
- 保護観察モードと呼ぶプログラムの新たな実行モードを導入し，暗黙的フローを制御する．

4.1 明示的フローの追跡

4.1.1 出力制限タグ

メモリやレジスタに格納されたデータがどの保護ポリシによる出力制限を受けるかを識別するために，メモリやレジスタを拡張し，出力制限タグを付加する．出力制限タグは，プログラムが保護ポリシの設定されたリソースからデータを読む際に OS がセットし，タグの内容をプログラムが自由に変更することはできない．

以下，出力制限タグがセットされたデータを出力制限データ，それ以外のデータを非出力制限データと呼ぶ．なお，非出力制限データの出力制限タグは 0 とする．

メモリ上のデータの出力制限タグは，ページテーブルに似た階層構造のタグテーブルによって管理することを想定しているが，本稿ではタグ管理の詳細については議論しない．

4.1.2 出力制限タグ伝搬

明示的フローはデータフローに付随して発生するため，データフローと同様に，命令セットアーキテクチャにおける一つの命令の実行において完結する．そこで，各命令の実行時に，命令のソースからディスティネーションに対して出力制限タグを伝搬させることによって，明示的フローを追跡する．出力制限タグの伝搬は論理和 (OR) 演算によって行い，ソースの中に一つでも出力制限データがあれば，ディスティネーションは出力制限データとなる．

代表的な RISC 命令を例に，タグ伝搬の様子を表 1 に示す．ただし，r1 はレジスタ r1 の出力制限タグを表し，[r1]

表 1　各命令実行時の出力制限タグ伝搬

命令	命令内容	出力制限タグ伝搬
ADD r1, r2, r3	r1 = r2 + r3	r1 = r2 OR r3
AND r1, r2, r3	r1 = r2 & r3	r1 = r2 OR r3
LOAD r1, r2	r1 = [r2]	r1 = [r2] OR r2
STORE r1, r2	[r2] = r1	r2 = [r1] OR r1

表 2　保護観察モードへの移行条件

命令	命令内容	移行条件
BRANCH r1, offset	if(r1) PC = PC + offset	r1 != 0
JMP r1	PC = r1	r1 != 0

図 4　保護観察モードステータス

はアドレス r1 が示すメモリの値を表す．LOAD，STORE 命令ではアドレスからも出力制限タグが伝搬する．

出力制限データがシステムコールを介してプログラム外部に出力される際，その出力が出力制限タグの示す保護ポリシに違反しないかどうかを OS が判断する．これによって明示的フローを介した情報漏洩が防止される．

4.2　暗黙的フローを介した情報漏洩の防止に向けたアプローチ

3.3 で述べたように，暗黙的フローをプログラム実行時に正確に追跡することはできない．しかし，暗黙的フローの発生は，条件分岐を行ってから，その分岐が合流するまでの期間に限定される．そこで，暗黙的フローを介した情報漏洩を以下のような方針で防止する．

- 出力制限データを起点とする暗黙的フローが発生する期間を区別し，その間の実行モードを保護観察モードとする．
- 保護観察モード中のプログラム外部とのインタラクションを保護ポリシに基づいて制限することで，保護観察モード中の情報漏洩を防止する．詳細は 4.4 で述べる．
- 保護観察モード中に発生した暗黙的フローの影響が保護観察モード終了後に及ばないようにすることで，保護観察モード終了後の情報漏洩を防止する．詳細は 4.5 で述べる．

4.3　保護観察モードへの移行および終了

4.3.1　保護観察モードへの移行

保護観察モードへは，出力制限データに依存した条件分岐命令ないし制御移行命令の実行によって移行する．代表的な RISC 命令を例に，保護観察モードへの移行条件を表 2 に示す．ただし，r1 はレジスタ r1 の出力制限タグを表し，PC はプログラムカウンタを表す．

保護観察モードへ移行するとき，プロセッサ内の保護観察モードステータス (図 4) に出力制限タグの値 (表 2 の例であれば，r1) を格納する．保護観察モードステータスのタグは，保護観察モードがどの保護ポリシによる制限を受けるものであるかを示す．保護観察モード中の制限については次節で述べる．

保護観察モードへの移行はコントロールフロー単位で行うため，マルチスレッドプログラムでは，スレッド切り替え時に保護観察モードステータスも切り替える．

4.3.2　保護観察モードの終了

保護観察モードは，保護観察モードに移行した際の分岐が合流することで終了する．しかし，通常，プロセッサが分岐の合流を動的に検出することはできない．そこでプログラマに，特定のアドレスを保護観察モードの終了アドレスとして申告させることを考える．申告はシステムコールを介して行われ，終了アドレスは保護観察モードステータスに格納する．

終了アドレスはプログラマによって申告されるが，プログラマが悪意をもって虚偽のアドレスを申告したとしても問題はない．それは，保護観察モードが以下の三つの性質をもつためである．

- 保護観察モード中にどのような実行パスを通ったとしても，プログラムカウンタが終了アドレスに到達するまで保護観察モードは続行する．
- 保護観察モード中の終了アドレスの申告，変更を禁止する．
- 保護観察モード終了後は，それまでの実行パスにかかわらず，同じ命令列が実行される．

このことからも明らかなように，プログラマが申告する終了アドレスは正確に分岐の合流ポイントである必要はなく，合流ポイント以降の任意のアドレスでかまわない．また，次節で述べる保護観察モード中の制限がプログラム機能の実現上問題にならない場合には，必ずしも保護観察モードを終了する必要はない．

4.4　システムコール呼び出しの制限

出力制限データに依存した分岐によって移行する保護観察モード中は，実行パスそのものが出力制限データに依存している．そのため，保護観察モード中のプログラム外部とのインタラクションは，出力制限データに関する情報の漏洩につながる．特に，プログラム外部へのデータ出力は，それが非出力制限データであったとしても，その出力データから実行パスが特定されることで，出力制限データの情報が漏洩する恐れがある．

データ出力を含むプログラム外部とのインタラクションは，システムコールを介して行われる．そこで，保護観察モード中の情報漏洩を防止するために，保護観察モード中のシステムコール呼び出しを保護ポリシに基づいて制限する．この制限は，保護観察モードステータスの出力制限タグの値を用いて，OS が行う．

Linux のいくつかのシステムコールを例に，システムコール呼び出しの制限条件を表 3 に示す．たとえば，保護ポリシがネットワークへの出力を禁止するものであった場合，保護観察モード中はすべての send が禁止される．同様に，特定のデバイスドライバへの出力のみ許可する保護ポリシであった場合には，そのデバイスドライバへの ioctl 以外，すべての write, send, mkdir 等が禁止される．

4.5　暗黙的フローの制御

保護観察モード中は，出力制限データを起点とする以

表 3　システムコールの呼び出し制限

システムコール	システムコール内容	システムコールの制限条件
write	ファイルディスクリプタへの出力	平文でのディスクへの出力が禁止されている場合
send	ネットワークソケットへの出力	ネットワークへの出力が禁止されている場合
ioctl	デバイスの制御	該当するデバイスへの出力が禁止されている場合
mkdir	ディレクトリの作成	平文でのディスクへの出力が禁止されている場合

下の二種類の暗黙的フローが発生する．

- 終点が出力制限データである暗黙的フロー
- 終点が非出力制限データである暗黙的フロー

出力制限データは通常モード時 (保護観察モード終了後) も出力制限を受けるため，前者の暗黙的フローを認識できなくとも，情報漏洩を防止する上で問題にはならない．一方，後者の暗黙的フローは，出力制限データの情報が非出力制限データに伝わるため，通常モード時に出力制限データの情報が漏洩する原因となる．

この問題に対して単純には，保護観察モード中に実行するすべての命令のデスティネーションに対して，出力制限タグを付加すればよいように思われる．しかし，3.3 で述べたように，命令を実行しないパスを通った場合にも暗黙的フローは発生するため，この方法はうまくいかない．

本節では，終点が非出力制限データである暗黙的フローの制御手法として，保護観察違反検出方式とロールバック方式の二つの異なる方式を説明する．

4.5.1　保護観察違反検出方式

出力制限タグを拡張して保護観察ビットを付加し，以下のような保護観察違反検出方式を考える．

- 保護観察モード中に非出力制限データを変更した場合，保護観察ビットを立てる．
- 通常モード時に保護観察ビットの立ったデータを使用した場合，保護観察違反としてプログラムの実行を停止する．
- 保護観察ビットは通常モード時にデータが上書きされた際に下げる．

保護観察ビットの導入により，保護観察モード中に変更した非出力制限データ，すなわち暗黙的フローの終点となった非出力制限データは，通常モード時に読むことができなくなる．これによって，保護観察モード中に発生した暗黙的フローの影響が保護観察モード終了後に及ぶことを防止する．

保護観察違反検出方式の有効性

保護観察違反検出方式では，命令を実行しないパスを通った時には保護観察ビットが立たない，という問題がある．しかし，この問題があっても本方式がなお有効であることを説明する．

図 5 のコードは，32 ビットの出力制限データ x を 1 ビットずつ漏洩させることを意図したものである．保護観察違反検出方式のもとでこのコードを実行した場合，x が 32 ビットの値を等確率でとるとすれば，1 ビットの試行 (printf の実行) のたびに $\frac{1}{2}$ の確率で保護観察違反が発生する．そのため，保護観察違反を起こす前に漏洩できる平均ビット数は，以下の式が示すようにわずか 2 ビットである．

```
y = 0;
for( i = 0; i < 32; i++){
   mask = 1 << i;
   if( x & mask ) y = 1;
   printf("%d", y);
}
```

図 5　出力制限データ x を 1 ビットずつ漏洩させるコード

```
y = 0;
for( i = 0; i <= 0xffffffff; i++){
   if( x == i ) y = 1;
   printf("%d", y);
}
```

図 6　出力制限データ x を総当たりで調べるコード

$$\lim_{n\to\infty}\sum_{k=1}^{n}(\frac{1}{2})^{k}k = 2$$

一方，図 6 に示すコードを実行した場合，標準出力に出力される 0 の数によって，x の値を漏洩させることができる．しかし，このコードの実行には，漏らしたいビット数 (今回の場合は 32) に対して幾何級数的な時間および出力データ量を必要とする (今回の場合は 2^{32})．そのため，現実的な時間で多くのビット数のデータが漏らすことは困難であり，実用上の問題は小さい．また，保護観察違反によって停止したプログラムは以後保護ポリシ付きのファイルにアクセスさせない，といったアクセス制限との組み合わせも有効である．

4.5.2　ロールバック方式

ロールバック方式では以下のような手法をとる．

- 保護観察モード中に非出力制限データを変更する場合，変更前の値を複製，保存する．
- 保護観察モード終了時に，変更された非出力制限データの値を変更前の値に置き換える (ロールバック)．

ロールバック方式では，保護観察モード終了時のロールバックによって，保護観察モード中に行った非出力制限データに対する変更が無効化され，保護観察モードに入る前の状態に戻る．これは，保護観察モード時の実行パスによらず，保護観察モード中に発生した暗黙的フローが無効化されることを意味する．そのため，保護観察違反検出方式とは異なり，暗黙的フローによる情報漏洩を完全に防止することができる．

ロールバック方式は，保護観察違反検出方式に比べ，メモリの複製とロールバックによるオーバーヘッドがかかる．メモリの複製は OS がページ単位で行い，ロールバックも OS が行うことを想定しており，この二つのオーバーヘッドは保護観察モード中に変更するメモリ

ページ数に依存する．オーバーヘッドの具体的な大きさに関しては今後評価していく予定である．また，マルチスレッドプログラムをロールバック方式で実行する場合，通常モードスレッドに対しては，複製した変更前のメモリページにアクセスさせる機構が必要となる．

4.6 プログラマへのインターフェース

保護観察違反検出方式およびロールバック方式では共に，保護観察モード中に変更した非出力制限データの値を保護観察モード終了後に読むことができない．これは，保護観察モード中のみ使用する自動変数などでは問題にならないが，グローバル変数など，保護観察モード中に値を変更し，保護観察モード終了後もその値を使用したい場合に問題となる．

そこで，プログラマへのインターフェースとして，特定のメモリ領域に出力制限タグを付加するためのシステムコール (settag) を提供する．settag システムコールを用いて，グローバル変数の格納場所などに自ら出力制限タグを付加することで，プログラムはその領域のデータを保護観察モード終了後も使用できるようになる．

5. 関連研究

本章では，情報漏洩の防止を目的とした研究の中で，特にインフォメーションフローに着目した研究について述べる．

専用言語で記述されたプログラムのインフォメーションフローを解析する研究がある[3]．Myers による JFlow[4] はそのような専用言語の一つであり，Java を拡張したものとなっている．専用言語ベースの解析では，ソースコードを静的に解析し，プログラマの決めたポリシに違反するインフォメーションフロー (明示的フローおよび暗黙的フロー) を検出する．これにより，ポリシに違反しないプログラムの開発が可能となる．しかし，このポリシはプログラマによって決定される固定的なものであり，データ所有者の意図を反映させることは難しく，情報を漏洩するか否かはプログラマの信頼性に依拠してしまう．

Vachharajani らによる RIFLE[7] は，バイナリ変換と専用プロセッサを組み合わせたインフォメーションフローの追跡手法である．バイナリ変換では，プログラムバイナリを静的に解析し，暗黙的フローを明示的フローに変換するコードと明示的フローを追跡するコードを追加する．変換後のバイナリを専用プロセッサで実行することでインフォメーションフローが追跡される．RIFLE はバイナリを解析の対象としているため，保護ポリシの決定においてプログラマの影響は排除され，データ所有者指向の情報漏洩防止が実現される．しかし，バイナリの解析はメモリ依存解析を必要とし，精度の高い解析は困難である．解析精度の低さはプログラム出力の過剰な制限につながり，ユーザがプログラム本来の機能を利用できない恐れがある．

Qin らによる LIFT[5] は，バイナリ変換を行うが専用プロセッサを必要としない手法であり，情報漏洩を含む幅広い脆弱性の克服を目的としている．しかし，暗黙的フローを考慮していないため，情報漏洩の防止機能は十分でない．

6. おわりに

本稿では，プログラム実行時のインフォメーションフローを動的に追跡，制御するプラットフォームによって，情報漏洩の防止が可能であることを示し，具体的なインフォメーションフローの追跡・制御手法を提案した．提案手法は，プログラムの信頼性に依存することなく情報漏洩の防止を可能にし，プログラマへのインターフェースを提供することで，プログラム機能の柔軟性も維持する．我々は現在，システムレベルエミュレータ Bochs[1] を用いて提案手法の実装を行っている．今後は実装したシステムを用いて，性能やメモリ容量に対するオーバーヘッドの評価，およびその抑制手法の検討を行うほか，プログラマやユーザに対するインターフェースの整備を行う予定である．

謝　辞

本研究は，一部 21 世紀 COE「情報技術戦略コア」，及び科学技術振興機構 CREST「ディペンダブル情報基盤」による．

参考文献

1) Bochs: the Open Source IA-32 Emulation Project, http://bochs.sourceforge.net.
2) Ku, W. and Chi, C.-H.: Survey on the Technological Aspects of Digital Rights Management., *ISC*, pp. 391–403 (2004).
3) Myers, A. and Sabelfeld, A.: Language-Based Information-Flow Security, *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 1, pp. 5–19 (2003).
4) Myers, A. C.: JFlow: Practical Mostly-Static Information Flow Control, *Proc. 26th ACM Symposium on Principles of Programming Languages* (1999).
5) Qin, F., Wang, C., Li, Z., seop Kim, H., Zhou, Y. and Wu, Y.: LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks., *Proceedings of the 39th International Symposium on Microarchitecture (MICRO)*, pp. 135–148 (2006).
6) TPM: Trusted Computing Group(TCG).
7) Vachharajani, N., Bridges, M. J., Chang, J., Rangan, R., Ottoni, G., Blome, J. A., Reis, G. A., Vachharajani, M. and August, D. I.: RIFLE: An Architectural Framework for User-Centric Information-Flow Security, *Proceedings of the 37th International Symposium on Microarchitecture (MICRO)* (2004).