情報処理学会論文誌 Vol.57 No.12 2836–2848 (Dec. 2016)

推薦論文

コンパイラを用いた情報フロー制御による情報漏洩防止機構

奥野 航平[1,†1,a)] 内匠 真也[1,†2] 大月 勇人[1,†3] 瀧本 栄二[1] 毛利 公一[1]

受付日 2015年11月9日, 採録日 2016年9月6日

概要：情報漏洩事件の多くは，人為的なミスを要因として発生していることが報告されている．そこで，本論文では，人為的なミスによる情報漏洩を防止するための機構 User-mode DF-Salvia を提案する．本機構は，情報フロー制御によってデータの利用方法を制限し，ユーザが意図しない情報の出力処理を制御することで情報漏洩を防止する．出力処理は，情報の源となったファイルに関連付いた保護ポリシに基づいて制御される．出力時の情報の保護ポリシを特定するためには，動的テイント解析を用いた情報フローの動的な追跡を利用する．これらのアクセス制御に必要な機能は，コンパイラによるコード変換を用いることでアプリケーションへ直接追加することによって実現し，プロセス単体でのアクセス制御を実現する．これにより，アプリケーションの置換え作業のみで本システムを導入でき，導入コストを削減できる．本機構の検証には，インターネット上の実アプリケーションを使用し，それらで情報漏洩が防止できることを確認した．

キーワード：アクセス制御，コード変換，情報フロー制御，動的テイント解析

A Data Loss Prevention System By Compiler and Information Flow Control

Kohei Okuno[1,†1,a)] Shin-ya Takumi[1,†2] Yuto Otsuki[1,†3] Eiji Takimoto[1] Koichi Mouri[1]

Received: November 9, 2015, Accepted: September 6, 2016

Abstract: Many data loss incidents have reported to be caused by human error. This paper proposes a data loss prevention system caused by human error, that's called *User-mode DF-Salvia*. Our system breaks the procedures of human-mistaken data output and restricts data usage by information flow control. An output is controlled by a protection policy associated with a data source file. To get a protection policy when outputting the data, we use the dynamic information tracking technique as dynamic taint analysis. These features for access control are inserted to an application program by code transform in compiler. These applications make possible to control itself. Therefore, a user can simply deploy the access control system by replacement of an application and reduce deployment costs. Our system is evaluated by real applications published on the internet and we confirmed data loss prevetion.

Keywords: access control, code transform, information flow control, dynamic taint analysis

1. はじめに

情報システムの発展によって機密情報が電子化されるようになり，電子化された機密情報の漏洩事件が増加している．情報漏洩の主な原因として，JNSA 2013 年情報セキュリティインシデントに関する調査報告書 [1] では，次のものをあげている．

アプリケーションの誤操作　ファイルを誤って電子メールに添付し，送信する．

データの管理ミス　ファイルを紛失（誤削除）する．ファ

[1] 立命館大学
Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[†1] 現在，株式会社インターネットイニシアティブ
Presently with Internet Initiative Japan Inc.
[†2] 現在，株式会社東芝
Presently with TOSHIBA CORPORATION
[†3] 現在，NTT セキュアプラットフォーム研究所
Presently with NTT Secure Platform Laboratories
[a)] kokuno@asl.cs.ritsumei.ac.jp

本論文の内容は 2015 年 3 月の第 68 回 CSEC 研究発表会にて報告され，同研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

© 2016 Information Processing Society of Japan

イルが行方不明（誤移動）になる．

紛失・置忘れ ファイルを USB フラッシュメモリなどの外部記憶装置へ書き込み，置忘れなどによって紛失する．

バグ プログラムの誤った処理によってデータが流出する．

これらは，人為的ミスによって発生しているといえる．また，情報漏洩の原因の約 8 割を占めており，大きな問題となっている．

情報漏洩を防止するための既存のセキュリティ技術として，ユーザ認証によるアクセス制御やファイルの暗号化などが存在する．しかし，これらの技術を利用した場合においても，1 度プログラムに読み込まれたデータは利用方法に制限がないため，人為的ミスによる情報漏洩の防止は困難である．また，既存のアクセス制御を拡張したシステムとしては，SELinux [2] や TOMOYO Linux [3], [4] などの強制アクセス制御がある．これらは，情報の伝達を制御する情報フロー制御によって，データの利用方法に対して制御できるため，人為的ミスによる情報漏洩の防止を可能とする．しかし，これらの情報フロー制御は，データ入出力の順序といった事前に定義された状態遷移に基づいて情報フローを推定するため，実際のプロセス内部の情報フローに従って制御されるとは限らない．また，従来のアクセス制御で使用されていた ACL（Access Control List）やパーミッションなどのファイルに対して設定するポリシのほかに，多種類のポリシを必要とするため，データの保護を目的とする設定の記述が複雑化しやすい．

以上の背景から，人為的ミスによる情報漏洩を防止する機構 User-mode DF-Salvia（以下，Salvia）を提案する．本機構は，ネットワークや外部記憶媒体など，プロセスによるデータ入出力処理を制限するアクセス制御機構を持つ．本機構では，ユーザが事前に保護対象であるファイルに対して，その利用方法を示したポリシ（以下，保護ポリシと記す）を付加する．アクセス制御機構は，プロセス内部の情報フローに基づいて出力されるデータをその保護ポリシに基づいて情報の伝播範囲を制限する．これにより，ファイル内の機密情報が外部に出力されることを防ぎ，人為的なミスによる情報漏洩を未然に防止する．Salvia の特徴は，以下の 2 点である．

- 情報フローに基づくアクセス制御が可能
- プログラム単位での適用が可能

情報フローは，あるオブジェクトが持つ情報が別のオブジェクトに伝わるときの流れである．情報はビット列から人間が判断して読み取れる内容であるため，従来のデータがコピーされる過程の追跡では情報フローの網羅的な追跡が困難である．動的な情報フローの追跡を実現する方式として，ハードウェアを用いた方式 [5], [6]，バイナリ変換を用いた方式 [7]，コード変換を用いた方式 [8] など様々な方式がある．既存研究の多くは，動的テイント解析やデータフロー解析技術を基本としている．動的テイント解析は，メモリやレジスタが保持するビット列に対してタグを付与し，ビット列が他のメモリ領域にコピーされる際にタグの伝播処理を行う．データフロー解析技術は，プログラムによる変換処理や伝播過程に着目している．これらの技術では，データとなるビット列を対象としており，ビット列が意味する情報を意識していない．情報漏洩防止の観点から考えると，処理対象が保持する情報を考慮しなければ，制御漏れや過剰制御が発生する可能性が高まる．したがって，データが持つ情報に着目した情報フローに基づくアクセス制御が，情報漏洩を防止するうえで不可欠であると考えられる．

既存研究の実現アプローチに関して，ハードウェアを用いた方式では，特殊なハードウェアを必要とするため，導入のハードルが高い．また，ハードウェアを制御するための OS も必要となり，コストが大きい．バイナリ変換を用いた方式は特殊なハードウェアを必要とせず，特定の実行可能ファイルのみを対象とする導入が容易である．コード変換を用いた方式は，ソースコードを専用のコンパイラにより再コンパイルすることでアクセス制御やセキュリティ機能を付与できるため，バイナリ変換方式と同様に導入が容易である．しかし，既存研究は，それぞれの文献でも言及しているように，暗黙的情報フローについて考慮していない．したがって，暗黙的情報フローによる情報漏洩を防止するという観点において，既存研究をそのまま適用することはできない．

Salvia は，コード変換を用いて単一プログラムに情報フロー追跡機能を付加し，特別な仕組みを必要としない方式をとる．コード変換は，ソースコードに情報フローを追跡するための変換を施すことで情報フローの追跡を実現する．情報フローの追跡は動的テイント解析を応用し，アクセス制御とタグ管理に必要な機能はライブラリとして提供される．Salvia では，情報フローに変化が発生する箇所とアクセス制御を行う箇所に，ライブラリを呼び出すコードを追加することで，動的なアクセス制御を実現する．データ変換や暗黙的情報フローの追跡を可能にするため，Salvia ではデータ変換が行われる関数や暗黙的情報フローを含む関数ごとに，アノテーションと呼ばれるプログラム開発者が提供する補助情報を用いる．アノテーションには，暗黙的情報フロー発生時のタグ伝播に関する情報や，データ変換による情報フローの状態変化に関する情報が関数単位で指定されている．

さらに，コード変換と，アクセス制御とタグ管理のライブラリ化によって，情報漏洩の危険が想定されたプログラムのみに適用することも可能となる．これらは，アクセス制御の導入コスト削減，システム全体のスループット低下抑制，プラットフォームに依存しないアクセス制御の実現といったメリットがある．

© 2016 Information Processing Society of Japan

以下，本論文では，2 章で Salvia について述べ，3 章で Salvia が提供する情報フローに基づくアクセス制御について述べる．4 章では，実アプリケーションを用いた評価としてアクセス制御機構の動作検証と性能評価について述べ，5 章で考察する．6 章では，関連研究について述べ，最後に 7 章で本論文をまとめる．

2. User-mode DF-Salvia

2.1 データ保護を実現するアクセス制御

図 1 に示すように，Salvia では保護したいファイルに対して保護ポリシをあらかじめ付与しておく．保護の対象をファイルとしているのは，コピーやメールへの添付といったユーザによる操作の多くがファイルを単位としているためであり，また情報がファイルという形で管理されるためである．保護ポリシには，ファイルへの書き出しの禁止（コピー禁止）やネットワークへの送信の禁止といった記述が可能である．したがって，Salvia は，ユーザの不注意によるメールへの誤添付やファイルの誤削除などの操作を防ぐことができる．すなわち，1 章であげた 4 つの原因のうち，アプリケーションの誤操作を防ぐことが可能になる．

保護ポリシの付与は，保護対象のファイルの管理責任を負う管理者によって行われる．Salvia では，管理者が必要なファイルごとに保護ポリシを設定する．保護ポリシをファイル単位で設定し，かつユーザによる保護ポリシの変更を防ぐため，Salvia は拡張ファイル属性を利用している．管理者が保護ポリシ設定ツールを用いてファイルごとに保護ポリシを設定すると，当該ツールによって保護ポリシがバイナリ形式で拡張ファイル属性に保存される．

Salvia を適用したアプリケーションには，保護ポリシに基づくアクセス制御を実現するアクセス制御機構が組み込まれる．アクセス制御機構の役割は，次のとおりである．

- ファイルオープン時に保護ポリシの有無を確認する．保護ポリシが付与されたファイルである場合は，その保護ポリシを解釈し，かつファイルと保護ポリシを関連付ける．保護ポリシが付与されていないファイルに対しては，何も行わない．
- 保護ポリシ付きファイルから情報を入力する場合は，保護ポリシに基づいてその入力処理を制御する．入力が拒否の場合は入力処理の結果がエラーとなり，入力が許可されている場合は，入力された情報とファイル，保護ポリシを関連付ける．
- 関連づく保護ポリシを持つ情報の出力時は，その保護ポリシに基づいた出力制御を入力時と同様に行う．

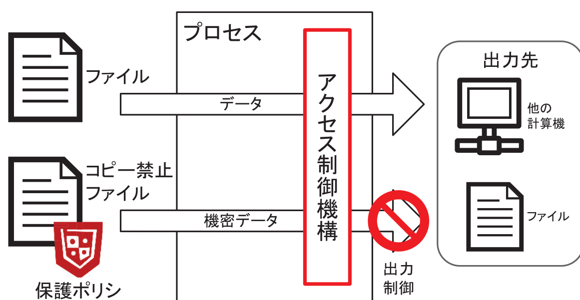

図 1 アクセス制御の概要

Fig. 1 Overview of access control.

Salvia では，アプリケーション開発を担当する開発者，ファイルの保護ポリシを設定する管理者，およびアプリケーションのユーザの 3 種類のロールを想定している．開発者は，2.3 節で後述する方法によってアプリケーション開発を行う．サードパーティ製アプリケーションのソースコードが入手可能であれば，アプリケーションを直接開発しなくても Salvia の適用が可能である．管理者は，ファイルに含まれる情報に応じた保護ポリシをファイルに付与する設定する役割を持つ．なお，本論文では，保護ポリシの設定における設定ミスがないものとして扱う．ユーザは，開発者が提供する Salvia 適用済みアプリケーションを操作し，管理者によって適切に保護ポリシが設定されたファイルの情報を利用する．ただし，Salvia は，1 章でも述べたとおり，ユーザのミスによる情報漏洩を防止するものであり，悪意あるユーザによる恣意的な情報漏洩を対象としていない．

2.2 動的な情報フローの追跡

2.1 節で述べたアクセス制御は，プロセスの情報フローに基づき，出力されようとしている情報の源となったファイルから保護ポリシを特定することで実現できる．しかし，情報は人間がデータの内容を理解して得られる内容であるため，データに含まれる情報量を機械的に判断することが困難である．そのため，データの変換をともなう情報フローや暗黙的な情報フローの追跡は困難である．特に，情報フローは，ユーザの入力操作や制御フローにより動的に変化する．したがって，情報フローに基づいたアクセス制御を行うためには，動的に変化する情報フローを適切に追跡することが必要となる．そこで，Salvia は，この課題を動的テイント解析の応用とコンパイラによるコード変換を用いて解決する．

動的テイント解析では，実行中のプロセスが保持するデータに対して識別子（タグ）を割り当てる．動的テイント解析は，データコピーなどによりデータが別のデータ領域へ伝播すると，当該データに割り当てられていたタグも同時に伝播させる仕組みを持つ．この伝播処理を繰り返すことで，出力されるデータには，オリジナルデータと同一のタグが割り当てられるようになる．Salvia では，タグと保護ポリシを対応させることで，アクセス制御に使用する保護ポリシの識別を可能とする．

ただし，データを対象とした従来の動的テイント解析を応用した情報フローの追跡は，データがコピーされるよう

© 2016 Information Processing Society of Japan

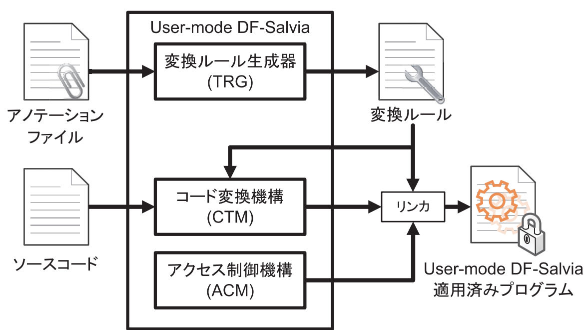

図 2 Salvia の構成
Fig. 2 Structure of Salvia.

な確実に情報が伝達する場合のみ有効である．これは，従来の動的テイント解析技術が演算などによってデータが持つ情報量の変化を考慮していないためである．データの変換をともなう情報フローや暗黙的な情報フローは，アノテーションを用いたプログラマによる情報フローの明示によって追跡を可能とする．

以上のことから，Salvia はコード変換を利用することにより，プラットフォームに依存せずプログラム単体でのアクセス制御を提供する．そのため，Salvia は，システムの更新を必要とせず，アクセス制御を必要とするアプリケーションにのみ導入することができるという利点がある．

2.3 構成

Salvia の構成を図 **2** に示す．Salvia は，コード変換機構（Code Transform Module; CTM），アクセス制御機構（Access Control Module; ACM），変換ルール生成器（Transform Rule Generator; TRG）の 3 つのコンポーネントから構成される．

ACM は，動的テイント解析を利用した情報フロー追跡に必要となるタグ管理機能と，データ出力を制御するアクセス制御機能を提供するライブラリである．タグ管理機能は，データ読込み時のタグ付与，情報伝播時のタグ伝播などを行う．アクセス制御機能は，データ入出力時に呼び出され，当該データに付与されたタグから保護ポリシを特定し，その入出力処理の可否判定を行う．

CTM は，プログラムのコードを変換するコンパイラである．CTM でコンパイルを行うと，ACM によって提供される各機能を適宜呼び出すためのコードがプログラム中に自動挿入される．

情報フローに基づくアクセス制御を実現するためには，明示的情報フローだけでなく暗黙的情報フローにも対応する必要がある．しかし，暗黙的情報フローの追跡は，コンパイル時における機械的な解析のみで実現することが困難である．また，ライブラリのようにソースコードからコンパイルされないものについても，情報フローを追跡することが困難である．

そこで，Salvia は，3.3 節で述べるアノテーションファイルを用いる．アノテーションファイルは，ソースコードからのコンパイルを行わないライブラリ関数や暗黙的情報フローを追跡するために必要な情報を記述したものである．TRG は，このアノテーションファイルを解析し，ライブラリ関数，データ変換，暗黙的情報フローの追跡のための変換ルールを生成するコンパイラである．変換ルールは，主に CTM がコード変換する際に利用される．さらに，実行状態に依存する情報フローについては，ACM が動的に使用するため，実行ファイルにリンクされ，ACM からの利用ができるようになっている．

3. 情報フローに基づくアクセス制御

3.1 ACM によるアクセス制御

ACM は，情報フロー追跡のためのタグ管理機能と，データ入出力に対するアクセス制御機能を提供するライブラリである．ACM は，これらの機能を提供するために，**表 1** に列挙されたアクセス制御用関数を提供する．

Salvia が提供するアクセス制御は，入出力データの源となったファイルに付与された保護ポリシに基づいて行われる．したがって，ACM がアクセス制御を行うためには，ファイルディスクリプタや FILE 構造体などといったファイル，およびソケットディスクリプタなどを識別するための識別子に関する情報が必要となる．以下，本論文では，当該識別子を FID（File ID）と表す．

ファイルオープン時は，`open_file()` によって ACM に通知が行われる．このとき，ACM は，当該ファイルに対して FID の付与，保護ポリシの取得，および FID と保護ポリシの対応付けを行う．ファイルに対する入力操作の際には，保護ポリシに基づいた入力処理の可否判定処理が行わる．可否判定は，`check_read()` を呼び出すことで行われる．ACM は，引数で指定された FID に対応する保護ポリシに基づいて，入力処理の可否判定結果を返す．判定の結果，不可であった場合は当該処理の返り値が強制的にエラー値となり，可であった場合は入力処理が行われる．

入力処理が終了すると，`tag_set()` によって ACM に入力データに関する情報を通知する．このとき，ACM は入力データが格納されたメモリ領域に対して情報フロー追跡のためのタグが付与する．付与するタグの決定は，FID に対する `tag_set()` が初めて呼び出された時点で ACM によって行われる．このとき，新規に割り当てられたタグと，FID，保護ポリシが対応付けられる．すなわち，タグの割当ては FID ごとに行われるため，保護ポリシを持つ複数のファイルが開かれた場合でも，各々の情報フローを適切に追跡することができる．

プログラムの実行にともない，データの伝播が発生すると，`tag_copy()` によって伝播先のメモリ領域に対応したタグ伝播が行われる．データがファイルやソケットに対し

© 2016 Information Processing Society of Japan

表 1 アクセス制御用関数の一覧
Table 1 Function list for access control.

種類	関数名	役割
ファイル管理	open_file(FID, FilePath)	FID として FilePath のファイルが開かれたことを通知する．
	open_socket(FID, IPAddr)	FID として IPAddr で示された IP アドレスに接続するためのソケットが開かれたことを通知する．
	dup_file(TargetFID, DupFID)	TargetFID の複製が DupFID として作成されたことを通知する．
	new_alias(TargetFID, LinkedFID)	TargetFID への別名が LinkedFID となっていることを通知する．
	close_file(FID)	FID が閉じられたことを通知する．
アクセス判定	check_read(FID)	FID の読み出し権限をチェックする．
	check_write(FID, Addr)	Addr に格納されたデータの FID に対する書き込み権限をチェックする．
タグ伝播	tag_set(Addr, Type, Len, FID)	Addr から Len バイトまでの領域に対して FID に対応したタグを付与する．
	tag_copy(DestAddr, SrcAddr, Type, Len)	SrcAddr から Len バイトまでの領域に付いたタグを DestAddr から Len バイトまでの領域にコピーする．
	tag_unset(Addr, Len)	Addr から Len バイトまでの領域に付いたタグを削除する．
関数呼び出し	caller_begin(Num, Addr, Len, ...)	関数の実引数の数（Num）と，各引数のアドレスとサイズを ACM に通知する．
	callee_begin(Addr, ...)	仮引数の Addr と caller_begin() で通知された実引数の情報を対応付ける．
	callee_end(Addr)	返り値を保持する変数のアドレス Addr を ACM に通知する．
	caller_end(Addr)	返り値が代入された変数のアドレス Addr を ACM に通知する．

て出力される際には，check_write() によって出力可否判定を行う．check_write() では，データに付与されたタグから保護ポリシを特定し，得られた保護ポリシに基づいて判定が行われる．その後，判定結果に基づいた処理が，入力処理と同じ要領で行われる．

このように，ACM は，提供する関数が適宜呼ばれることによって，タグ管理と入出力制御の可否判定を行う．ただし，ACM はライブラリであるため，その機能を有効に使うためには，CTM によるコード変換が必要である．

3.2 コード変換

CTM の主な役割は，情報フローを追跡し，アクセス制御を行うために，適宜 ACM を呼び出すコードを追加することである．また，ACM のアクセス制御に対する役割は，データ入出力制御の実行可否を判定するのみであるため，判定結果によって異なる制御が行われるようなコードの追加も行う．コード変換は，コンパイル時に行われ，以下の処理を対象とする．

- 代入処理
- 関数呼び出し（引数・戻り値を通じた情報フロー）

代入処理は，プログラム上の変数を別の変数にコピーする処理であり，情報フローが発生する．CTM は，代入処理の後にタグを伝播させるための関数 tag_copy() を追加する．代入処理に演算を含む場合は，データが変換されているため，情報が変わったと判断し，タグを伝播させない．また，定数でデータが上書きされた場合は，上書き処理の後に tag_unset() を追加してタグを削除する．

関数呼び出しでは，関数の引数が暗黙的にコピーされる．また，呼び出し元に戻るときには，戻り値による情報フローが発生する．これらを追跡するために，CTM は，関数を呼び出すコードの前後および関数の出入り口に ACM を呼び出すコードを追加する．また，追加されたコードによって ACM は，タグを以下の手順で伝播させる．

(1) 関数呼び出し前に，実引数のアドレスとそのサイズを caller_begin() によって ACM に通知する．
(2) 関数の入口で，仮引数のアドレスとそのサイズを callee_begin() によって ACM に通知し，ACM が手順 (1) で得た情報を基にタグをコピーする．
(3) 関数の出口で，戻り値として返される変数のアドレスとそのサイズを callee_end() によって ACM に通知する．
(4) 関数呼び出し元で戻り値が代入される前に，代入先のアドレスとそのサイズを caller_end() によって ACM に通知し，ACM が手順 (3) で得た情報を基にタグをコピーする．

3.3 アノテーション

CTM が単独で提供できるコード変換機能には，限界がある．たとえば，glibc のようなライブラリが提供する関数はソースコードからコンパイルを行わないため，関数利用時に行うべきコード変換を行うことができない．また，暗黙的情報フローの検出は機械的に行うことが困難であるため，やはりコード変換を適切に行うことが困難である．Salvia では，このような問題に対して，アノテーションを用いることで解決を図っている．アノテーションは，C 言語をベースとした Salvia 独自のアノテーション記述言語で記述される．Salvia では，アノテーションが記述されたファイルをアノテーションファイルと呼称し，プログラム開発者によって提供されるものとする．以下，本節ではアノテーションと TRG による変換ルールについて述べる．

© 2016 Information Processing Society of Japan

3.3.1 アノテーションの対象

アノテーションは，関数を単位として記述される．ただし，すべての関数について記述する必要はなく，以下に該当する関数について記述すればよい．

- FID が作成・削除される関数
- FID を通じてファイル入出力を行う関数
- 暗黙的情報フローを含む情報フローが発生する関数

上記の関数は，ライブラリ関数とプログラマが定義した関数に分けられる．ライブラリ関数は，情報フローが発生する関数であっても CTM によってコンパイルされていないため，明示する必要がある．なお，これらの関数は，プログラミング言語やプログラミングインタフェースの仕様書などに記載されており，仕様からアノテーションを作成できる．また，あらかじめ，ひな形としてアノテーションを作成しておくことで，同一のライブラリ関数を使用する場合に再利用が可能となり，アノテーションを作成するコストを削減できる．

プログラマが定義した関数については，データ変換や暗黙的情報フローが発生する場合に記述が必要となる．暗黙的情報フローは機械的な追跡が困難である．また，暗黙的情報フローの発生箇所すべてについて，適切な制御を行うことも困難である．そこで，Salvia では，暗黙的情報フローそのものを解析せず，暗黙的情報フローが含まれる関数を単位として対応する．これにより，暗黙的情報フローが発生する関数についてアノテーションを記述することによる対応が可能となる．

3.3.2 アノテーションの記述ルール

図 3 に，アノテーションの例として `fopen()`，`fgets()`，`memcpy()`，`memset()` のライブラリ関数に対するアノテーションを示す．アノテーションの中で使用する変数の型は，`typedef` を用いて宣言する（1–2 行目）．アノテーションは，それぞれの変換対象となる関数ごとに，`check`，`fail`，`post` の 3 種類のブロックに分かれている．それぞれのブロック内は，C 言語と同じ文法規則が利用できる．唯一，C 言語と異なる点は，アノテーションで指定された関数の戻り値を関数名で指定する点である．

`check` ブロックには，関数を実行する前に行う必要のある処理を記述する（13–15 行目）．このブロックには，暗黙的に 0 で初期化された変数 `permit` が宣言されており，この変数に対して 1 を設定することで，該当関数の実行が許可される．たとえば，`fgets()` の場合，ファイルの入力を許可することを確認する `check_read()` を呼び出し，アクセス可否の確認を行う（14 行目）．

`fail` ブロックには，アクセス判定で拒否されたときに，関数のエラーをエミュレーションするための処理を記述する（16–18 行目）．`check` ブロックが宣言されていた場合，このブロックは省略できず，必ず戻り値を設定する必要がある．`fgets` 関数の場合は，読み取りエラーで戻り値として 0 を返すため，その処理を記述する（17 行目）．

```
1  typedef struct _IO_FILE FILE;
2  typedef unsigned int size_t;
3
4  FILE *fopen(const char *path, const char *mode)
5  {
6          post (fopen != 0) {
7                  open_file(fopen, path);
8          }
9  }
10
11 int fgets(char *s, int size, FILE *stream)
12 {
13         check {
14                 permit = check_read(stream);
15         }
16         fail {
17                 fgets = 0;
18         }
19         post (fgets != 0) {
20                 int len = strlen(s);
21                 tag_set(s, LOSSLESS, len, stream);
22         }
23 }
24
25 void *memcpy(void *dest, void *src, size_t n)
26 {
27         post (1) {
28                 tag_copy(dest, src, LOSSLESS, n);
29         }
30 }
31
32 void *memset(void *s, int c, size_t n)
33 {
34         post (1) {
35                 int i;
36                 for (i = 0; i < n; i++) {
37                         tag_copy(s + i, &c, LOSSLESS, 1);
38                 }
39         }
40 }
```

図 3 アノテーションファイルの例
Fig. 3 Example of annotation file.

`post` ブロックには，関数を実行した後に行う必要のある処理を記述する（6–8，19–22，27–29，34–39 行目）．この処理は，`post` の後の括弧内で指定した条件を満たしたときのみ実行される．この条件には，通常，アノテーションで指定した関数が正常終了したときの条件を指定する．`fopen()` の場合は，正常にファイルを開いた後に `open_file()` を呼び出し，ファイルパスと FID を関連付ける処理を行う（7 行目）．`fgets()` の場合は，データ読み出し後に `tag_set()` を呼び出し，データのアドレスに対してタグを設定する処理を行う（21 行目）．`memcpy()`，`memset()` の場合は，データコピー後に `tag_copy()` を呼び出し，タグを伝播させる処理を行う（28，37 行目）．

3.3.3 アノテーションに基づくコード変換

CTM は，アノテーションが与えられていない関数呼び出しについて，3.2 節で述べた方法でコード変換を行う．ただし，アノテーションが与えられた関数呼び出しについては，アノテーションに従うために異なる方法でコード変換を行う．CTM は，アノテーションが与えられた関数に対して，その呼び出しを**図 4** に示すようなラッパ関数の呼び出しにすべて置き換えられる．

図 4 は，図 3 の `fgets()` に対するアノテーションから生成されたラッパ関数である．ラッパ関数とアノテーションは，3 行目と `check` ブロック，5–9 行目と `post` ブロック，11 行目と `fail` ブロックがそれぞれ対応している．

© 2016 Information Processing Society of Japan

```
1   char *acm_fgets(char *s, int size, FILE *stream)
2   {
3           int permit = check_read(stream);
4           if (permit) {
5                   ret = fgets(s, size, stream);
6                   if (ret != 0) {
7                           int len = strlen(s);
8                           taint_set(s, len, stream);
9                   }
10          } else {
11                  ret = 0;
12          }
13  }
```

図 4 fgets() 用のラッパ関数
Fig. 4 Wrapper function of fgets().

```
ret = pfunc(arg1, arg2, arg3);
```

⇩

```
1: if (dyn_prehook(pfunc, arg1, arg2, arg3)) {
2:   ret = pfunc(arg1, arg2, arg3);
3:   dyn_posthook(pfunc, ret, arg1, arg2, arg3);
4: } else {
5:   ret = dyn_error(pfunc);
6: }
```

図 5 間接参照を用いた関数呼び出しの変換
Fig. 5 Translation of function calls via indirect reference.

3.3.4 間接参照への対応

間接参照を用いた関数呼び出しでは，呼び出される関数が静的に決定しない．呼び出される関数を動的に取得するために，実行時において実際に呼び出された関数のアドレスに基づいた対応が必要となる．

CTM は，間接参照を用いた関数呼び出しに対して，**図 5** に示すようなコード変換を行う．追加された関数は，アノテーションの動作と対応しており，check ブロックの動作が 図 5 の 1 行目，post ブロックの動作が 3 行目，fail ブロックの動作が 5 行目に対応している．これらの関数は，ACM が提供するものである．

ACM は，上記の関数が呼ばれると，実際に呼び出される関数のアドレスに基づいた制御を，変換ルールを用いて行う．変換ルールには，プログラム中の各関数のアドレスとアノテーションが対応付けられているため，ACM は実際に呼ばれた関数のアドレスから，該当するアノテーションを取得できる．このとき，次項で述べる変換ルールに基づき，アノテーションで記述された各ブロックを実行する．該当するアノテーションが変換ルールに存在しない場合は，3.2 節で述べた関数呼び出しによる情報フローの追跡を適用する．

3.3.5 変換ルール

CTM は，アノテーションに基づいて，コード変換を行う．しかし，アノテーションファイルの記述に誤りがあると，エラーの要因となる．なおかつ，発生したエラーの原因がプログラム本体とアノテーションファイルのどちらにあるかを判別することが困難になる．

TRG は，アノテーションファイルに基づいて変換ルールを生成する．変換ルールは，独自の形式を持つオブジェクトファイルである．TRG は，アノテーションファイルを事前に解析することでアノテーションのエラーを検出する．

また，ACM がブロックごとに処理できるよう，変換ルールは独自のバイナリ形式に変換される．ACM はこのバイナリ処理するインタプリタを持っており，このインタプリタで処理することで変換ルールを動的に適用することができる．

表 2 データ変換行列
Table 2 Data conversion matrix.

		変換処理の分類			
		可逆	暗号	情報保持	不可逆
直前の分類	可逆	可逆	暗号	情報保持	不可逆
	暗号	暗号	暗号	情報保持	不可逆
	情報保持	情報保持	暗号	情報保持	不可逆
	不可逆	不可逆	不可逆	不可逆	不可逆

3.4 情報フローの分類

情報フローは，データの加工によって情報量が変動する．たとえば，ある情報を含んだデータのハッシュ値を取得した場合，そのハッシュ値自体に含まれる情報量は 0 ととらえることができる．一方，暗号化された場合は，暗号データに可逆性があるため，情報量自体の変化はないととらえることができる．情報量が 0 となった情報フローに対して出力制御を行うことは，過剰な制御につながる．一方で，減少したとしても何らかの情報を保持する情報フローに対しては，出力制御を行う必要がある．Salvia では，このような情報フローにおける情報量の変化に着目し，情報フローがとりうる状態および演算を次のように定義する．また，ある状態の情報フローに対して，情報フローの状態を遷移させる処理が行われた場合，処理後の情報フローの状態は**表 2** に基づいて決まる．

- 可逆（LOSSLESS）
  データの可逆性があり，かつ，情報を保持している情報フロー．例として，BASE64 などのエンコードがある．
- 暗号（CRYPT）
  追加情報を用いることでデータの可逆性を持ち，かつ，情報を保持している情報フロー．例として，AES，DES などの暗号処理がある．
- 情報保持（PRESERVE）
  データの可逆性は持たないが，情報を保持している情報フロー．例として，画像ファイルの不可逆圧縮や文字のレンダリング処理がある．
- 不可逆（LOSSY）
  情報が破壊される情報フロー．例として，MD5, SHA-1 などのハッシュ関数がある．

ただし，上記分類の境界は，明確でない場合がある．たとえば，ハッシュ関数でも MD4 は，ハッシュ値から元の

© 2016 Information Processing Society of Japan

情報を得ることが可能である．この点については，アプリケーションの開発者が，情報フローの変化が発生する処理をどういう意図で行っているかに基づいて，分類を指定する必要がある．

Salvia では，情報保護の観点から，情報が完全に破壊された不可逆以外の情報フローは，すべてアクセス制御の対象とする．同じ観点から，情報量が低下した情報保持状態の情報フローに対して，情報保持となる処理を行った場合は，情報量が完全に破壊されたとはいえないため，情報保持の状態を維持する．

表 1 にあげた ACM が提供する関数には，引数として `Type` を指定するものがある．Type には，上であげた 4 つの状態を指定する．ACM はタグに 4 つの状態を含めており，情報フローの状態まで追跡できるようになっている．

4. 評価

本章では，Salvia の実アプリケーションに対する機能検証と性能評価について述べる．評価に用いた PC は，Intel Core i5-2320 の CPU と 16 GB のメモリを搭載し，OS として Fedora 20 x86_64 (Linux 3.17.7) が動作している．また，CTM は，LLVM 3.4 を使用して実装したものを使用した．

4.1 機能検証

4.1.1 検証方法

機能検証では，2 つの観点から検証を行った．1 つは，Salvia が想定するユーザの誤操作が行われたと仮定し，その誤操作による情報漏洩を防げることを確認するためである．もう 1 つは，Salvia が暗黙的情報フローを正しく追跡し，最終的に保護ポリシに従ったアクセス制御が行われることを確認するためである．

これらの観点に基づき，ユーザが実際にファイルに対して操作する状況を考慮し，以下に示す 5 つのアプリケーションを用いた．

- cp コマンド
  保護対象のファイルに対するファイルコピー操作のみを防止し，非保護対象のファイルコピー操作が正しく行えることを確認する．
- ftp コマンド
  保護対象のファイルの送信を防止し，非保護対象のファイルの送信が正しく行えることを確認する．
- Miniperl（Perl インタプリタ）
  Perl インタプリタをコンパイルすることで Perl スクリプトに対してアクセス制御が適用できることを確認する．これは，CGI（Common Gateway Interface）を想定し，Web ブラウザ経由でのファイル送信なども Salvia によって制御できることを確認する．検証では，以下に示す 2 種類の Perl スクリプトを動作させる．
  – ファイルコピー
    ファイルから 1 行ずつ読み出し，ファイルへ書き込む．
  – CGI スクリプト
    指定したファイルからデータを読み出し，読み出した内容を含む Web ページを生成する．
- Mailx（メーラ）
  保護対象のファイルをメールで添付して送信できないことを確認する．また，メール添付時の処理は暗黙的情報フローを含むため，Salvia の情報フロー追跡能力検証の意味合いも持つ．
- nkf（文字コード変換プログラム）
  文字コード変換による暗黙的情報フローを追跡できることを検証する．文字コード変換自体は，汎用的な機能であることから，検証で用いることとした．

検証時に使用するアノテーションは，C ライブラリ用に作成したものを用いる．また，上記アプリケーションのうち，Mailx と nkf は暗黙的情報フローを含む関数を持つ．Mailx には，バイナリファイルの添付時に `ctob64()` による BASE64 のエンコード処理がある．検証では，この `ctob64()` 用のアノテーションを定義し，エンコード後のデータにタグを伝播させるアノテーションを追加で定義した．nkf には，異なる文字コードへ変換するときにテーブルを使用したデータの変換がある．Mailx と同様に，変換処理である `iconv()` と `oconv()` に対してアノテーションを追加で定義した．

保護対象のファイルには，ネットワークおよびファイルへの出力禁止となる保護ポリシを与えた．非保護対象のファイルには保護ポリシの設定を行っていない．検証は，上記アプリケーションのアクセス制御によって，保護ポリシに従った制御が行われたか否かで判定する．具体的には，保護対象のファイルと非保護対象のファイルとをそれぞれ上記アプリケーションで操作し，ファイルから読み込んだデータの出力時に，保護対象のファイルに基づくデータのみ出力が抑制され，非保護対象のファイルに基づくデータの出力は実行されることを確認する．

4.1.2 検証結果

検証の結果，すべてのアプリケーションを Salvia でコンパイルでき，かつ，それぞれのアプリケーションにおいて保護ポリシに従ったアクセス制御が行われたことを確認した．cp コマンドは，`read()`，`write()` システムコールによってファイルをコピーするため，C ライブラリ用のアノテーションで制御が可能であった．ftp コマンドは，間接参照を使用し関数呼び出しがあり，アノテーションの動的な適用によって制御が可能であった．Miniperl は，代入処理によって文字列をコピーする処理と関数の戻り値によってデータを受け渡す処理があり，それぞれ代入処理と関数呼び出しによる情報フローの追跡によって制御が可能で

© 2016 Information Processing Society of Japan

表 3　UnixBench によるベンチマーク結果
Table 3　Benchmark result by UnixBench.

テスト項目名	通常版	Salvia 版	低下率（%）
Dhrystone 2 using register variables	2938.1	4.2	99.9
Double-Precision Whetstone	761.1	64.4	91.5
Execl Throughput	792.2	246	68.9
File Copy 1024 bufsize 2000 maxblocks	1471	581.8	60.4
File Copy 256 bufsize 500 maxblocks	920.9	357.9	61.1
File Copy 4096 bufsize 8000 maxblocks	2982.8	1367.7	54.1
Process Creation	1176.3	405.3	65.5
Shell Scripts (1 concurrent)	2255.3	2227.9	1.21
Shell Scripts (8 concurrent)	4856.6	4826.2	0.63
System Call Overhead	1721.8	147.5	91.4

あった．

Mailx の検証では，バイナリファイルを添付した場合においても，`ctob64()` のアノテーションによって，保護ポリシに従った制御を確認した．nkf についても，文字コード変換処理において，保護ポリシに従った制御を確認した．

Mailx と nkf において正しく保護ポリシに従ったアクセス制御が行われたことは，Salvia が暗黙的な情報フローを正しく追跡できたことを意味する．すなわち，暗黙的情報フローの追跡を対象外としている既存研究 [5], [6], [7], [8] に対して，Salvia が優位性を持つことが明らかとなった．

以上の結果より，Salvia によるアノテーションを用いた情報フローの追跡によって，保護ポリシに従ったアクセス制御が適切に行われたことを確認した．

4.2　ベンチマークテスト

4.2.1　方法

本評価では，Salvia によるコード変換が，アプリケーション性能に与える影響について確認する．具体的には，通常の LLVM でコンパイルした UnixBench（以下，通常版）と Salvia を適用した UnixBench（以下，Salvia 版）をそれぞれ実行し，各テストに対応するスコアを比較する．ただし，現行の Salvia はマルチプロセスやマルチスレッドで動作するプログラムに対応していない．そのため，本評価では Pipe Throughput と Pipe-based Context Switching，およびマルチスレッドで行われるテスト項目を除外した．

4.2.2　テスト結果

実行結果である UnixBench のスコアと，通常版に対する Salvia 版の低下率を**表 3** に示す．Dhrystone 2 using register variables，Double-Precision Whetstone，System Call Overhead は，それぞれ 99.9%，91.5%，91.4%と著しく性能が低下した．これらのテストプログラムにはコード変換対象である関数呼び出しや代入処理が数多く含まれ，かつそれらの実行回数が多かったことから，ACM の呼び出し回数増加によるオーバヘッドが性能低下の原因と考えられる．

File Copy では，54–60%程度の性能低下が確認できる．ファイルのコピー処理は，同一バッファを利用するためタグ伝播に関する処理の実行頻度は高くなかった．一方で，ファイル入出力の実行回数が多かった．したがって，File Copy における性能低下の大きな要因は，ファイル入出力時におけるアクセス制御に関するオーバヘッドであったと考えられる．アクセス制御時は，データに付与されたタグに関連付けられたファイルの保護ポリシ取得および確認が行われる．この処理においてキャッシュを用いることで，ポリシ取得に関するオーバヘッドを改善できると考えられる．

Execl Throughput と Process Creation は，同一プロセスの複製を行う．複製時には，実行ファイルがメモリ上にロードされた後，ACM の初期化処理が行われるため，性能が低下したと考える．

Shell Scripts の性能低下は，1%程度と小さい．この理由は，スコアの計測対象がシェルスクリプトであり，計測用プログラムがほとんど動作しないためである．

5.　考察

5.1　Salvia の有効性

機能性検証の結果より，Salvia は，ファイルに対する入出力処理をファイルに付与された保護ポリシに基づいてアクセス制御できることが明らかとなった．特に，暗黙的情報フローを含む Mailx と nkf についても，保護ポリシに従った制御が行われたことを確認できた．したがって，既存研究では追跡が困難であった暗黙的情報フローを含むアプリケーションに対して Salvia を適用できるため，適用範囲が広く有効性も高いと考えられる．

Salvia のオーバヘッドの観点では，マイクロベンチマークより I/O バウンドな処理よりも CPU バウンドな処理による性能低下が大きかった．実アプリケーションとして個人情報をファイルで管理・編集するものを考えた場合，CPU 演算量が少ないため数倍程度のオーバヘッドに収まることが期待できる．一方，個人情報から種々の統計やマ

© 2016 Information Processing Society of Japan

イニング処理を施す場合は，数百倍程度のオーバヘッドになると予想される．ただし，統計やマイニングは，演算によって情報が変化していくため，タグ伝播処理の効率化やアノテーションでの指定により，オーバヘッドの軽減が可能であると考えられる．

Salvia 自体は，単一プロセスに対してのみ有効な手段であり，複数プロセスにわたる情報フロー追跡には対応していない．この点については，プロセス間通信などを利用して保護ポリシやタグに関する情報を交換することで対応できると考えられ，今後検討していく．逆に，OS やハードウェアなど，システム全体を置き換える必要がなく，情報漏洩が起こりうるアプリケーションのみに対する適用が可能であるため，オーバヘッドの増加を局所化し，導入コストを抑えることができる．

Salvia は，コンパイラによるコード変換，ACM によるアクセス制御を行う．アクセス制御の対象はアプリケーションの入出力要求に対するものであり，保護ポリシに違反した入出力要求に対して，エラーを返す．アプリケーションからは，Salvia における処理は，通常の入出力エラーと同等である．すなわち，プログラムの振舞いに影響を与えることなく，保護ポリシに従ったアクセス制御を提供することができる．

5.2 制約と限界

Salvia を利用するためには，ソースコードが必要である．自社開発システムやオープンソースソフトウェアを利用する場合は特に問題ないが，プロプライエタリなアプリケーションに対して適用することができない．また，ソースコードを基に作成するアノテーションファイルは，開発者にとって負担となる．ただし，標準ライブラリなどの一般的に利用されるライブラリ用のアノテーションファイルは，いったん作成すれば他のアプリケーション開発時にも再利用できる．

現行の Salvia が対応できるアプリケーションは，機能検証で用いたアプリケーションのように比較的単純な構成である必要がある．より複雑な情報フローの追跡を行うためには，Salvia の拡張が必要である．たとえば，LLVM IR の命令セットを網羅し，プログラムの複雑さに関係なく明示的な情報フロー追跡に対応する必要がある．また，暗黙的情報フローについては，アノテーションの記法を拡張する必要がある．現行のアノテーション記法では，1 つの関数内で完結する暗黙的情報フローに対応している．そのため，複数の関数にまたがるなど複雑な暗黙的情報フローに対応した記法の拡張が必要であり，今後検討していく必要がある．

Salvia は，人為的な誤操作によって発生する情報漏洩を防ぐものである．また，その目的を実現するためには，保護対象のファイルについて，適切に保護ポリシを設定し，かつ Salvia を適用したアプリケーションで処理することが必要となる．したがって，管理者による保護ポリシの設定ミスや，悪意あるユーザによる Salvia を適用していないアプリケーションでの恣意的な情報漏洩については，防止することができない．保護ポリシの誤設定については，その設定をサポートするツールの開発によって軽減することができる．悪意あるユーザによる情報漏洩に対しては，Salvia だけでなく SELinux などを併用することで対処できると考えられる．たとえば，SELinux が提供する Type Enforcement を利用し，保護ポリシが付与されたファイルを Salvia 以外からアクセスできないようにすることが有効である．

6. 関連研究

6.1 先行研究との位置付け

著者らは先行研究として，OS ベースの透過的な方法で情報漏洩を防止するシステムである Salvia Linux [9]，および DF-Slavia [10] を開発してきた．Salvia Linux は，OS による透過的な方法で情報漏洩を防止するシステムである．保護対象のファイルを開いたプロセスが行う入出力要求に対してアクセス制御を行うため，その制御単位がプロセスであり粒度が粗いという課題があった．DF-Salvia は，Salvia Linux と同様にアクセス制御を OS によって行うが，コンパイル時のデータフロー解析結果を利用することで細粒度のデータフローをアクセス制御の単位とすることを可能にしたシステムである．

これらの先行研究はアクセス制御を OS が担うため，OS を含めたシステム導入が必要となる．また，アクセス制御によるオーバヘッドが OS で発生するため，システム全体への影響が大きくなる．情報漏洩の観点については，DF-Salvia でデータフローを単位として制御が可能となったが，より抽象度が高い情報フローへの対応が必要である．

本論文で提案した Salvia は，システム導入のコスト削減のためアプリケーション単位での導入が可能になること，アクセス制御によるオーバヘッドが Salvia 適用アプリケーションで閉じられることから，アプリケーション単位で導入できる方法を採用した．また，アノテーションの利用により，明示的情報フローと暗黙的情報フローの両方を追跡可能にしている．

6.2 動的テイント解析

動的テイント解析は，外部から攻撃の検知 [5] やマルウェアの解析 [6] など悪意あるデータを追跡するために使用されることがある．Argos [5] は，ゼロデイ攻撃を検知するためのエミュレータである．川古谷らは，マルウェアによって書き換えられたデータを追跡する手法 [6] を提案している．そのほかにも，libdft [7] や DTA++ [11] などの汎用的に動的テイント解析を利用できるようにしたフレームワー

© 2016 Information Processing Society of Japan

クがある．これらの手法は，データに対して汚染（テイント）をマークしているが，そのデータが具体的にどのファイル，ネットワークアドレスから入力されたかを区別しない．したがって，テイントタグを付与されたデータすべてに対して画一的な制御が行われる．一方，Salvia は，データソースであるファイルの保護ポリシ，データ，タグをそれぞれ紐付けて管理する．したがって，異なる保護ポリシを持つ複数のファイルを同時に扱う場合であっても，Salvia は各データに紐付けられたファイルの保護ポリシにに応じてアクセス制御を実現することができる．

一般に，暗黙的な情報フローが発生すると，タグの伝播が発生しないため情報フローを追跡できない [12]．そのため，意図的に暗黙的な情報フローを発生させた場合，タグが伝播しない問題が発生する．川古谷らは，マルウェアのコードが書き換えたデータに対してもタグを付けることで問題を解決している．

吉濱らは Java を対象とし，JavaVM のプログラムカウンタにもタグを付与することで制御構造を通じた暗黙的情報フローの追跡を可能としている [13]．しかし，暗黙的情報フローを含む関数には対応しておらず，静的解析の必要性を論じている．DTA++ では，事前にプログラムを解析することで，暗黙的な情報フローの追跡を実現している．Salvia では，アノテーションを用いて暗黙的な情報フローの追跡を可能とし，情報フローによって出力されたデータに対して情報が含まれているかどうかを分類して追跡する．そのため，暗黙的な情報フローが発生したときでも，アノテーションを適切に定義することで，過剰な制御を抑制することができる．アノテーションは，ソースプログラムに基づいて作成されるため，幅広い種類の情報フローの追跡を可能とさせている．また，アノテーションに関する技術は，他の研究にも応用が可能であり，たとえば文献 [13] の研究に応用することで，既存の研究にも貢献が期待できる．

6.3 Usage Control

既存のセキュリティモデルとして，Usage Control（UCON）[14] が提案されている．UCON は，データへのアクセス許可後の利用を制御するセキュリティモデルであり，情報漏洩の防止や知的財産の管理（DRM）などに適用が可能である．UCON で使用されるポリシとしては，「コピー禁止」，「30 日後にファイルを削除」，「5 回まで動画像を利用可能」などが利用できる．システムの実装としては，VMM と OS を使用した方式 [15] や Android 上に実装したシステム [16] などがある．

本手法では，情報漏洩の防止をターゲットとし，ポリシには，プロセスの動作を記述する方式を採用している．ポリシは，情報フローの追跡によってデータに対応付けられるため，データに応じてプロセスの動作を制限できる．ただし，プロセスが動作していないときのデータ利用は制限できず，主として DRM の用途において，自動的にファイルを削除するといった受動的なイベントには本システムで対応できない．しかし，人為的ミスによる情報漏洩は，プロセスが動作しているときに発生するため，プロセスの能動的な動作を制限するだけで十分な対策ができるといえる．

また，利用者の個人情報を扱うためのモデルとして，文献 [17] の研究がある．当該モデルでは，個人情報を提供する利用者は，個人情報がどのように処理されるかを示す保護ポリシを選択する．サービス提供者は，利用者が指定した保護ポリシに基づいて個人情報が処理されるよう，個人情報処理プログラムを変換する．個人情報の処理として，個人情報を個人が特定できないデータに変化することが例としてあげられている．このように，当該モデルは，個人情報を提供する利用者が，サービスを安心して利用することができるようにする枠組みである．なお，当該モデルの実装では，暗黙的情報フローの追跡に対応していない．

2.1 節でも述べたように，Salvia は，個人情報を含む機密情報を管理する側において，人為的なミスによる情報漏洩を防止するためのシステムである．文献 [17] では，個人情報をどのように利用者が満足する方法で処理するかを主眼としている．したがって，サービス提供者の過失による情報漏洩を防ぐことはできない．一方，Salvia は，機密情報をどのように適切に扱うかを主眼としているため，情報漏洩を防ぐことができる．このように，両者の枠組みはそれぞれ情報の提供者からの視点と，管理および使用する側の視点といった形で異なる．

7. おわりに

本論文では，人為的なミスによる情報漏洩を防止する User-mode DF-Salvia について述べた．Salvia は，データの利用方法をユーザが保護ポリシで指定でき，情報フローに基づいてデータの利用方法を制限することで，情報漏洩の防止を可能とする．また，コンパイラを用いたコード変換により，情報フローの追跡，および，アクセス制御をプログラム単体で実現した．コード変換を実現したことで，既存のアプリケーションをコンパイルするだけでアクセス制御を導入きるメリットがある．評価では，実アプリケーションとして Mailx と nkf を用いて動作を検証した．性能面では Salvia を適用していない状態と比較して，I/O バウンドな処理で 60%程度，CPU バウンドな処理で 90%を超える性能低下となった．このことから，実用的なシステムを構築するうえで，オーバヘッドの削減が重要である．

今後の課題としては，アノテーションの記述漏れを防ぐようにするための手法の考案や，実行時のオーバヘッド削減用の静的解析などがある．

© 2016 Information Processing Society of Japan

参考文献

[1] NPO日本ネットワークセキュリティ協会（JNSA）：JNSA 2013年情報セキュリティインシデントに関する調査報告書，入手先 ⟨http://www.jnsa.org/result/incident/⟩ (2015).

[2] Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System, *Proc. FREENIX Track: 2001 USENIX Annual Technical Conference*, pp.29–42 (2001).

[3] 原田季栄，保理江高志，田中一男：TOMOYO Linux—タスク構造体の拡張によるセキュリティ強化Linux，*Proc. Linux Conference 2004*, pp.1–10 (2004).

[4] 原田季栄，半田哲夫，橋本正樹，田中英彦：アプリケーションの実行状況に基づく強制アクセス制御方式，情報処理学会論文誌，Vol.53, No.9, pp.2130–2147 (2012).

[5] Portokalidis, G., Slowinska, A. and Bos, H.: Argos: An Emulator for Fingerprinting Zero-Day Attacks for advertised honeypots with automatic signature generation, *Proc. 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pp.15–27 (2006).

[6] 川古谷裕平，岩村　誠，針生剛男：テイント伝搬に基づく解析対象コードの追跡方法，情報処理学会論文誌，Vol.54, No.8, pp.2079–2089 (2013).

[7] Kemerlis, V.P., Portokalidis, G., Jee, K. and Keromytis, A.D.: Libdft: Practical Dynamic Data Flow Tracking for Commodity Systems, *Proc. 8th ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments 2012*, pp.121–132 (2012).

[8] Chang, W., Streiff, B. and Lin, C.: Efficient and Extensible Security Enforcement Using Dynamic Data Flow Analysis, *Proc. 15th ACM Conference on Computer and Communications Security 2008*, pp.39–50 (2008).

[9] 鈴来和久，一柳淑美，毛利公一，大久保英嗣：Privacy-Aware OS Salviaにおけるデータアクセス時のコンテキストに基づく適応的データ保護方式，情報処理学会論文誌：コンピューティングシステム，Vol.47, No.SIG3 (ACS 13), pp.1–15 (2006).

[10] 内匠真也，奥野航平，大月勇人，瀧本栄二，毛利公一：コンパイラとOSの連携によるデータフロー追跡手法，情報処理学会論文誌，Vol.56, No.12, pp.2313–2323 (2015).

[11] McCamant, M.G.K.S. and Song, P.P.D.: DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation, *Proc. 18th Annual Network and Distributed System Security Symposium* (2011).

[12] Cavallaro, L., Saxena, P. and Sekar, R.: On the Limits of Information Flow Techniques for Malware Analysis and Containment, *Proc. 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Vol.5352, pp.143–163 (2008).

[13] 吉濱佐知子，工藤道治，小柳和子：動的アプローチによる言語ベースの情報フロー制御，情報処理学会論文誌，Vol.48, No.9, pp.3060–3072 (2007).

[14] Park, J. and Sandhu, R.: The UCONABC Usage Control Model, *ACM Trans. Information and System Security*, Vol.7, No.1, pp.128–174 (2004).

[15] Xu, M., Jiang, X., Sandhu, R. and Zhang, X.: Towards a VMM-based Usage Control Framework for OS Kernel Integrity Protection, *Proc. 12th ACM Symposium on Access Control Models and Technologies 2007*, pp.71–80 (2007).

[16] Bai, G., Gu, L., Feng, T., Guo, Y. and Chen, X.: Context-Aware Usage Control for Android, *Security and Privacy in Communication Networks*, Vol.50, pp.326–343 (2010).

[17] Kuto, K., Takahashi, K., Kawamura, T. and Sugahara, K.: Program Conversion for the Protection of Personal Informaion, *Proc. TSP2013*, pp.1599–1604 (2013).

推薦文

実効性の追求に特化した論文で，よく整理され論文誌化できるほどにまとまった成果になっている．また，実験による提案手法の評価も行っている．

（コンピュータセキュリティ研究会主査　鳥居　悟）

奥野 航平

1990年生．2013年立命館大学情報理工学部情報システム学科卒業，2015年同大学大学院理工学研究科博士前期課程情報理工学専攻修了，同年株式会社インターネットイニシアティブ入社，現在に至る．クラウド基盤構築・運用に従事．

内匠 真也　（正会員）

1990年生．2013年立命館大学情報理工学部情報システム学科卒業，2015年同大学大学院理工学研究科博士前期課程情報理工学専攻修了，同年（株）東芝研究開発センター入社，現在に至る．主にシステムソフトウェアの研究開発に従事．

大月 勇人　（正会員）

1988年生．2011年立命館大学情報理工学部情報システム学科卒業，2013年同大学大学院理工学研究科博士前期課程情報理工学専攻修了．2016年同大学院情報理工学研究科博士後期課程情報理工学専攻修了，同年日本電信電話株式会社入社，現在，NTTセキュアプラットフォーム研究所勤務．博士（工学）．マルウェア解析技術に関する研究に従事．

© 2016 Information Processing Society of Japan

情報処理学会論文誌 Vol.57 No.12 2836–2848 (Dec. 2016)

瀧本 栄二 （正会員）

1976年生．1999年立命館大学理工学部情報学科卒業，2001年同大学大学院理工学研究科博士前期課程修了，2005年同研究科博士後期課程単位取得退学，同年（株）ATR適応コミュニケーション研究所専任研究員，2010年立命館大学情報理工学部情報システム学科助手，現在に至る．博士（工学）．主にシステムソフトウェア，コンピュータセキュリティ，無線通信に関する研究に従事．

毛利 公一 （正会員）

1994年立命館大学理工学部情報工学科卒業，1996年同大学大学院理工学研究科修士課程情報システム学専攻修了，1999年同研究科博士課程後期課程総合理工学専攻修了．同年東京農工大学工学部情報コミュニケーション工学科助手，2002年立命館大学理工学部情報学科講師，2004年同大学情報理工学部情報システム学科講師，2008年同准教授，2014年同教授となり，現在に至る．博士（工学）．オペレーティングシステム，仮想化技術，コンピュータセキュリティ等の研究に従事．電子情報通信学会，ACM，IEEE-CS，USENIX各会員．

© 2016 Information Processing Society of Japan