Vol.2011-DPS-146 No.45
Vol.2011-CSEC-52 No.45
2011/3/11

データフロー情報に基づく アクセス制御のためのプログラム解析

樫山武浩[†1] 河島裕亮[†2] 井田章三[†2]
瀧本栄二[†3] 毛利公一[†3]

近年，頻発している情報漏洩の原因の多くは，誤操作や管理ミス，紛失といった正当なアクセス権限を持つユーザの過失によるものである．我々は，これらに起因する情報漏洩を防止するためのアクセス制御機構を備えたオペレーティングシステム DF-Salvia を開発してきた．DF-Salvia では，プロセスの挙動を監視し，プロセスがデータを書き出す際に，データの読み込み元のファイルに付加された保護ポリシに従って計算資源へのアクセス要求の実行可否を判定する．これにより，予め保護すべきデータが格納されるファイルにポリシを適用することで，正当なアクセス権限を持つユーザによるデータ漏洩を防止する．DF-Salvia のアクセス機構では，コンパイラにおいてプログラム解析したデータフロー情報に基づくことで，データの読み込み元のファイルを特定し，適用すべき保護ポリシを決定する．そこで，本稿では，DF-Salvia のアクセス制御のためのデータフロー情報のプログラム解析について述べる．

Program analysis for mandatory access control based on data flow information

Takehiro Kashiyama,[†1] Yusuke Kawashima,[†2]
Shozo Ida,[†2] Eiji Takimoto[†3] and Koichi Mouri[†3]

Recently, leak incidents of data have occurred frequently. In many cases, the factors of data leakage are as follows: taking data out illegally or unfairly, or misoperation by a user with authority to access them. We have been developing operating system DF-Salvia for the purpose of preventing data leakage which resulted from these factors. DF-Salvia monitors behavior of processes not to access computer resources which have a possibility of data leakage. When a process requests to access such resources, DF-Salvia allows the operation if it does not violate the data protection policy. The access control of DF-Salvia depends on data flow information which is obtained by program analysis. In this paper, method for analyzing the data flow information is described.

1. はじめに

近年，計算機が普及し，プライバシデータはデジタルデータとして保存，管理されている．これによって，作業の効率化や企業の提供するサービスの品質向上につながったが，同時に記憶媒体やネットワークを通じて個人情報が流出する事件が頻繁に発生している．個人情報の漏洩は，ユーザのプライバシ侵害につながるだけでなく，企業に対しイメージダウンや補償問題による金銭的な負担を与える．このような事態に対し，2005 年 4 月 1 日から企業や自治体に対する個人情報の利用と取扱いに関する義務や制限を定めた個人情報保護法[1)]が施行された．これにより，個人情報の取扱いに対する意識は高まっているが，依然として情報漏洩事件は発生している．

文献 2) では，2008 年に報道された個人情報漏洩事件の原因の調査について述べられており,「誤操作」,「盗難」,「管理ミス」,「紛失・置き忘れ」の比率が最も高いという結果が報告されている．具体的には,「誤操作」では顧客に他の個人の情報を電子メールに添付して送信してしまったという事例が多数報告されている．また,「盗難」,「管理ミス」,「紛失・置き忘れ」では，顧客などの個人情報を記憶媒体に保存し，外部に持ち出したことが原因となっている事例が多い．これらから，データ漏洩の多くが不当アクセスによるものではなく，正当なアクセス権限を持つユーザが原因となっていることが分かる．このようなデータ漏洩は，暗号化や認証などの外部からの攻撃を防ぐことを目的としたセキュリティ技術では防ぐことは難しい．

以上の背景より，正当なアクセス権限を持つユーザによるプライバシデータ漏洩を防止する技術の開発が必要である．そこで，我々はこれまで，プライバシデータの漏洩を防止することを目的として，強制アクセス機構を備えるオペレーティングシステム Salvia[3)] の開発を行ってきた．Salvia では，データ提供者の意図する保護方針をデータ保護ポリシ (以下，ポリシ) として定義し，保護対象のファイル (以下，保護ファイル) と組にして管理する．Salvia は，保護ファイルのデータ (以下，保護データ) を読み込んだプロセスに対し，

†1 立命館大学グローバル・イノベーション研究機構
Ritsumeikan Global Innovation Research Organization, Ritsumeikan University.
†2 立命館大学大学院理工学研究科
College of Information Science and Engineering, Ritsumeikan University.
†3 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University.

Vol.2011-DPS-146 No.45
Vol.2011-CSEC-52 No.45
2011/3/11

ポリシに基づいた強制アクセス制御を課すことにより，プライバシを考慮したデータ保護を実現している．ただし，Salvia のアクセス制御は，計算機資源へアクセスしたとき，当該プロセスが過去に読み込んだ全保護ファイルのポリシに違反しないときに限りアクセスを許可するため，プロセスに対して過剰にアクセスを制限してしまうことがある．

そこで，Salvia のアクセス制御の粒度を更に細かくし，データごとにより正確なアクセス制御を行うことを可能とする DF-Salvia を開発してきた．DF-Salvia は，Salvia がプロセスを主体としたアクセス制御を行うことに対し，プロセス内のデータフローを主体としたアクセス制御を行う．これにより，プロセスの計算機資源へ書き込まれるデータがどのファイルを起源としているのか，その対応付けを正確に識別してアクセス制御することが可能となる．すなわち，Salvia では，保護データに対する過剰なアクセス制限や，保護が不要なデータまで制限してしまう場合があるが，DF-Salvia ではそれらを防止することが可能となるためユーザビリティが向上する．

DF-Salvia では，データフローを主体としたアクセス制御のために，事前にプログラム解析したデータフロー情報を利用する．これまで，データフロー情報は，プログラムから手作業で生成してきた．ただし，データ漏洩を防止するソフトウェア基盤として，DF-Salvia を社会に導入するためには，データフロー情報の生成過程を自動化する必要がある．そこで，我々は，データフロー情報のプログラム解析機構を備えたセキュアコンパイラの開発を目指している．セキュアコンパイラが実現すれば，プログラムの生成から実行までの一貫した過程において，情報漏洩を防止するソフトウェア基盤を構築できる．本稿では，DF-Salvia のアクセス制御のためのデータフロー情報のプログラム解析について述べる．

2. DF-Salvia のアクセス制御

DF-Salvia では，OS はファイルを単位としてデータを管理する点に着目し，プライバシデータを含むファイルごとにポリシを設定する．そして，ファイル，パイプ，ソケット，子プロセスといった計算機資源に対するアクセス要求が発生した際，コンテキストとポリシに基づいて，その実行可否を判定する．DF-Salvia では，アクセス制御の主体をデータフロー単位とすることで，アクセス要求に課すべきポリシを正確に判別する．これにより，Salvia の課題であった過剰制御を軽減することができる．DF-Salvia におけるアクセス制御モデルを**図 1** に示す．また，アクセス制御の手順を以下に示す．

(1) 事前にプログラム解析したデータフロー情報に対してデータフロー ID を割り当てる．
(2) read システムコールが発行された時，読み込み対象が保護ファイルなら，読み込まれるデータが属するデータフロー ID に対して，対象保護ファイルのポリシを適用する．
(3) write システムコールが発行された時，書き出されるデータが属するデータフロー ID に対して適用されたポリシの有無を確認する．
(4) ポリシが適用されている場合は，そのポリシに従ってシステムコールの実行の可否を判断する．

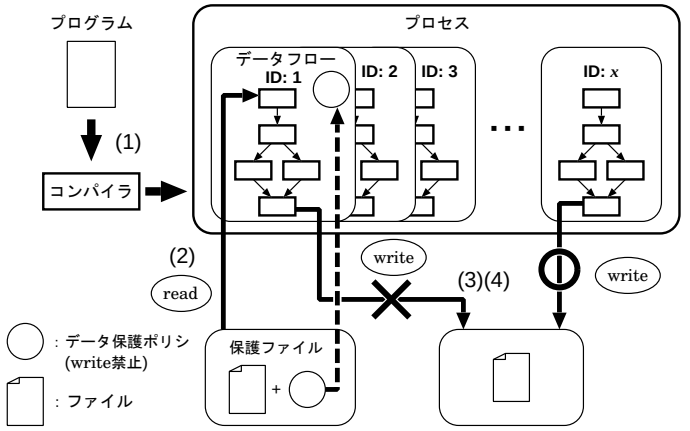

図 1 DF-Salvia のアクセス制御モデル
Fig. 1 Access control model of DF-Salvia.

上記の処理を行うためには，手順 (2)，(3) において，システムコールが発行された時点で使用されているデータが属するデータフロー ID を OS が特定できなければならない．データフロー ID は，システムコールの発行元であるライブラリ関数コールの命令アドレスから求める．事前にプログラム解析するデータフロー情報として，read システムコールを発行するライブラリ関数コールを定義点，write システムコールを発行するライブラリ関数コールを使用点とする定義-使用連鎖を利用する．また，それぞれの定義点，使用点について，ライブラリ関数コールの命令アドレスを求めておく．そのため，システムコールにおいて取得した命令アドレスとデータフロー情報に含まれる命令アドレスを比較することで，データフロー ID を特定できる．なお，OS 実行時のライブラリ関数コールの命令アドレスは，システムコール発行時にプロセスのスタックを解析することで求める．

Vol.2011-DPS-146 No.45
Vol.2011-CSEC-52 No.45
2011/3/11

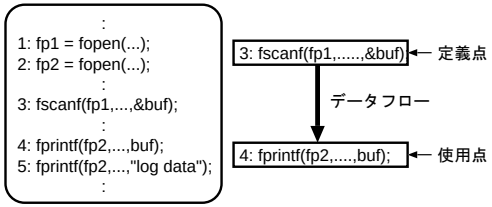

図 2 定義-使用連鎖
Fig. 2 Define-use chain.

3. アクセス制御のためのデータフロー情報

DF-Salvia では，事前にプログラム解析したデータフロー情報に基づくことで，アクセス要求時に適用するポリシを判別する．また，ポリシが不要となった場合，それを削除する契機をデータフロー情報から得る．以下に，DF-Salvia のアクセス制御で利用するデータフロー情報について述べる．

3.1 定義-使用連鎖

コンパイラでは，原始言語プログラムを解析して，目的プログラムを生成するが，一般的に，コードを速くサイズの小さいものとするコード最適化を行う[4]．コード最適化では，プログラム中の変数定義の流れ，すなわち，データフローを把握することが必要不可欠である．そこで，プログラム解析されるデータフロー情報のひとつが定義-使用連鎖である．DF-Salvia では，アクセス制御が基づくデータフロー情報として，定義-使用連鎖を利用する．コード最適化で使用される定義-使用連鎖は，変数への値の代入文を定義点とし，その変数が使用された文を使用点とする．一方，DF-Salvia のアクセス制御では，read システムコールの発行元となるライブラリ関数コールを定義点，write システムコールの発行元となるライブラリ関数コールを使用点とする定義-使用連鎖を利用する．DF-Salvia で利用する定義-使用連鎖を**図 2** に示す．定義点を read システムコールの発行元となるライブラリ関数コールに限定しているのは，これ以外の文でファイルのデータを格納した変数が定義されることはないからである．同様に，使用点を write システムコールの発行元となるライブラリ関数コールに限定しているのは，これ以外の文で情報漏洩の要因となる計算機資源へのデータの書き込みが行われないからである．

3.2 最終使用点

DF-Salvia では，ファイルからデータが読み込まれた場合，そのデータが属するデータフローにポリシを適用する．ポリシは，それが適用されたデータフローのアクセス制御にのみ必要なものである．したがって，当該データフローの処理が全て終了すれば，以降のアクセス制御に不必要となる．そこで，データフローの最後の処理が実行された後，データフローからポリシを削除する．しかし，DF-Salvia では，処理の実行後において，データフローからポリシを削除するかどうかを定義-使用連鎖から決定できない．これは，定義-使用連鎖が変数の定義文とその変数を使用した命令文の集合であり，それぞれの命令文の実行順などの制御フローに関わる情報を保持しないためである．そこで，定義-使用連鎖が表現するデータフローから最後に実行される命令文を解析し，データフロー情報に付加して DF-Salvia に提供する．以降，本稿では，データフローの最後に実行される命令文を最終使用点と呼ぶ．

4. データフロー情報のプログラム解析

本章では，まず，データフロー情報をプログラム解析するための基本手法について述べる．そして，データフロー情報に基づいたアクセス制御において発生する問題を挙げ，それを解決する手法を提案する．なお，本稿では，手続き間やグローバル変数については触れず，手続き内の局所変数のみを対象としたプログラム解析について述べる．

4.1 プログラム解析の基本手法

データフロー情報の解析には，まず，プログラムから制御フローを求める．そして，制御フローに基づいて，生存変数解析[4] を行う．生存変数解析は，プログラム中のある文で定義された変数が，以降の文で使用されるか否かを求める解析である．あるプログラム点において，変数が使用される場合を変数が生存していると呼び，そうでない場合を変数が死滅している呼ぶ．ブロック B で定義される変数の集合を $def[B]$，ブロック B で使用される変数の集合を $user[B]$，ブロック B の直前で生存している変数の集合を $IN[B]$，ブロック B の直後で生存している変数の集合を $OUT[B]$ とする．また，ブロック B の後続節をブロック S とした場合，データフロー方程式は，以下のように定義される．

$$IN[B] = user[B] \cup (OUT[B] - def[B]) \quad (1)$$

$$OUT[B] = \bigcup IN[S] \quad (2)$$

制御フローの各ブロックについて，$def[B]$ と $user[B]$ を求め，上記のデータフロー方程式を $IN[B]$ と $OUT[B]$ の値に変化がなくなるまで反復アルゴリズム[5] により解くことで，各プログラム点における生存変数を決定する．そして，定義文から各ブロックの $IN[B]$ と $OUT[B]$ の値に基づいて，制御フローをたどることで，定義-使用連鎖を生成する．ここで

Vol.2011-DPS-146 No.45
Vol.2011-CSEC-52 No.45
2011/3/11

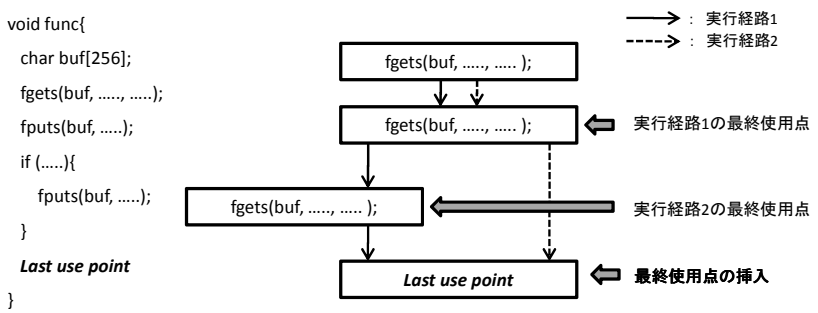

図 3 最終使用点の挿入
Fig. 3 Insertion of last use point.

述べる定義-使用連鎖は，以下の要素によって構成される．

- 定義点：変数の代入文，read システムコールの発行元となるライブラリ関数コール
- 使用点：定義点で値が代入された変数を使用する文の集合

定義-使用連鎖は，変数の定義からその変数が使用される文の集合を表現する．代入によって別の変数に値が格納された場合は，その代入文を定義点とした別の定義-使用連鎖を生成する．また，最終使用点は，定義点から制御フローの全ての実行経路をたどり，最後に使用される文を最終使用点として決定する．

4.2 アクセス制御のための拡張

DF-Salvia のアクセス制御において，アクセス要求に課すべきポリシを判別するためには，プログラム中で実行された命令文から一意なデータフローを決定する必要がある．制御フローが複数の実行経路を持つ場合や変数の再定義が行われるプログラムでは，複数のデータフローにおいて，実行経路上の命令文が共有される場合がある．その場合，命令文から一意なデータフローを決定できない．本節では，上記を踏まえ，各条件下のプログラムで発生する問題を示すとともに，それを解決するための手法について述べる．

4.2.1 変数の定義文が 1 つの場合

本項では，それぞれの変数の定義文がプログラム中で一つしか存在しない場合を検討する．このようなプログラムでは，使用点における命令文に必ず定義点の値が到達し，命令文から一意なデータフローが求められる．しかし，if 文や for 文により制御フローに複数の実行経路が存在するプログラムでは，定義-使用連鎖から最終使用点を決定できない場合がある．最終使用点は，プログラムの実行経路ごとに決定する．しかし，ある実行経路の最終使用点が別の途中経路上にある場合，それを最終使用点とすることはできない．つまり，最終使用点を決定できない経路が存在することになる．このような場合，**図 3** に示すように，プログラム中に新たな最終使用点を挿入する．具体的には，OS に対してポリシの削除を通知するシステムコールを予め作成しておき，その呼び出し命令をプログラムに挿入する．そして，その呼び出し命令を使用点として定義-使用連鎖に追加する．最終使用点は，以下の条件を満たす箇所に挿入する．

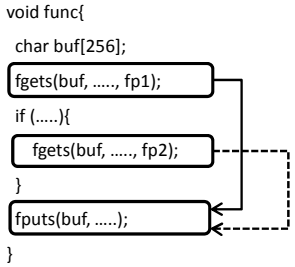

図 4 使用点の共有
Fig. 4 Sharing of use point.

- 後続の実行経路で，使用点が存在しないブロック
- 定義を含むブロックが後支配するブロック

上記に示したブロックは，定義点からの全ての実行経路で通過される．そのため，データフローに適用されたポリシを必ず削除することができる．

4.2.2 変数の定義文が複数存在する場合

本項では，プログラム中のある地点で定義され，その後の実行経路で再定義される変数が存在する場合を検討する．まず，プログラムの制御フローにおいて，1 つの実行経路しか存在しない場合，それぞれの使用点の命令文では，ある 1 つの定義点から値が到達し，命令文から一意なデータフローが求められる．一方，制御フローに複数の実行経路が存在するプログラムでは，**図 4** に示すように，使用点を共有する複数の定義-使用連鎖が生成される場合がある．この場合，命令文から複数のデータフローが求められるが，DF-Salvia では，どのデータフローに適用されるポリシをアクセス要求に課すべきかを決定できず，アクセス制御を実行できない．この問題については，前項で述べた最終使用点に加え，新たな最終使用点を挿入することで解決する．具体的には，**図 5** に示すように，定義点からの実行経路において，その変数の別の定義点が存在する場合，その直前に最終使用点を挿入する．これによ

Vol.2011-DPS-146 No.45
Vol.2011-CSEC-52 No.45
2011/3/11

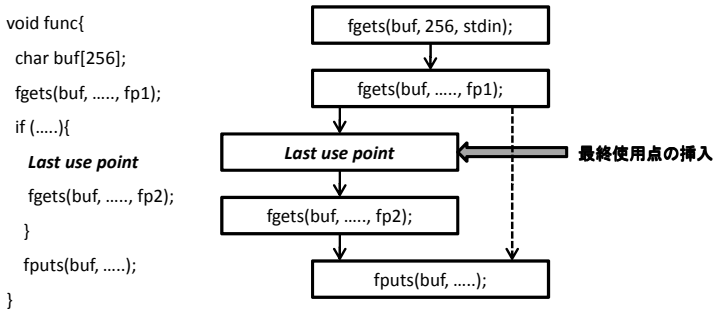

図 5 最終使用点の追加
Fig. 5 Addition of use point.

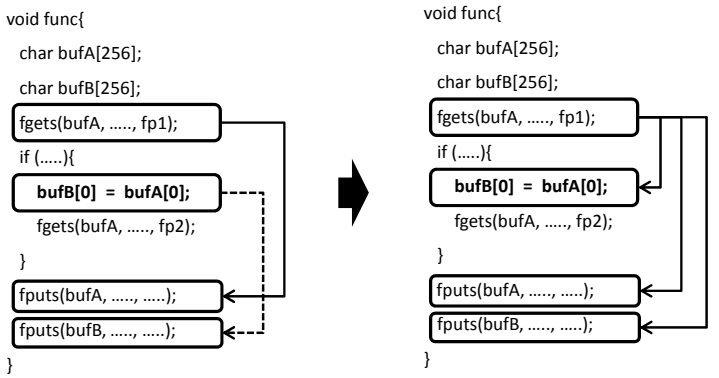

図 6 定義-使用連鎖の結合
Fig. 6 Combination of define-use chain.

り，データフローにポリシが設定される直前に，同じ変数の別のデータフローに適用されたポリシを削除できる．つまり，ある変数について，使用点を共有するデータフローに着目した場合，プログラムの実行時にそれらに同時にポリシが適用されることがない．そのため，DF-Salvia において，アクセス要求に適用すべき一意なポリシを決定できる．

4.2.3 変数から変数への代入文が存在する場合

代入文により保護データが変数に格納された場合，その代入文を定義点とするデータフローにもポリシを適用する必要がある．しかし，DF-Salvia では，ライブラリ関数コールから発行された read システムコールでのみデータフローにポリシを適用するために，代入文を定義点とするデータフローにはポリシが適用されない．したがって，使用点での処理でアクセス制御が行われない．この問題については，**図 6** に示すように，代入文を定義点とする定義-使用連鎖と，ライブラリ関数コールを定義点とする定義-使用連鎖を結合することで解決できる．これにより，代入文を定義点とするデータフローに対してもポリシに従ったアクセス制御が行われる．しかし，制御フローに複数の実行経路が存在するプログラムにおいて，使用点を共有する複数の定義-使用連鎖が生成される場合に，前項で示した内容と同様の問題が発生する．前項では，変数が再定義される直前に，最終使用点を挿入することで，この問題を解決する手法を述べた．前項の手法では，それぞれのデータフローが 1 つの変数を介したものであることを前提とした．しかし，代入文によって結合された定義-使用連鎖は，複数の変数を介したデータフローを表現する．そのため，図 6 の代入文とその直後の定義文の間に，前項の方法に従って最終使用点を挿入した場合，代入文を定義点とするデータフローの使用点に到達する前にポリシが削除され，アクセス制御が正常に行われない．そこで，代入文において，定義-使用連鎖を結合せず，代入元の変数のデータフローに適用されたポリシを代入先のデータフローに適用することを通知するシステムコールを用意し，**図 7** に示すように，代入文の直後に挿入する．このシステムコールを代入通知コールと呼ぶ．また，代入通知コールを使用点として定義-使用連鎖に追加する．代入通知コールが発行された場合，DF-Salvia において，以下の処理を実行する．

(1) 代入通知コールの所属するデータフローを決定する．
(2) （1）の定義点を使用点とするデータフローを決定する．
(3) （2）のデータフローにポリシが適用されている場合，そのポリシを（1）のデータフローに適用する．

上記の仕組みを導入することで，定義-使用連鎖を 1 つの変数を介したデータフローとした定義を崩さずに，代入文を定義点とするデータフローに対してポリシを適用できる．したがって，前項で述べた最終使用点の挿入方法を使用することで，使用点の共有により発生する問題を解決できる．

4.2.4 別名が存在する場合

本項では，プログラム中において，定義された変数の別名[6] を使用する文が存在する場合について検討する．別名の変数についても代入と同様に，別の定義-使用連鎖として解析される．また，変数間の別名関係は，代入文で生成されるため，それを接続点として定義-使

Vol.2011-DPS-146 No.45
Vol.2011-CSEC-52 No.45
2011/3/11

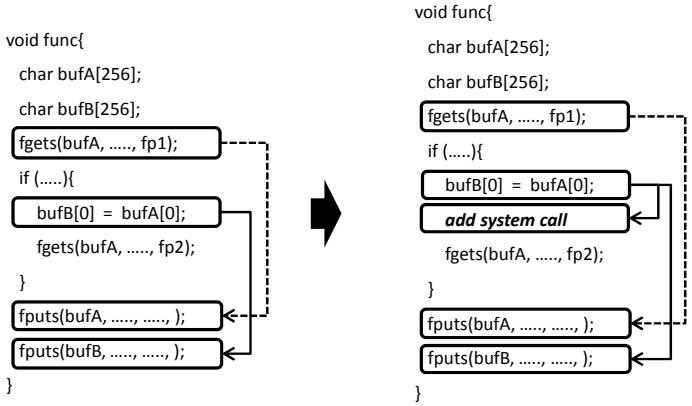

図 7 代入通知コールの挿入
Fig. 7 Insertion of system call for assignment statement.

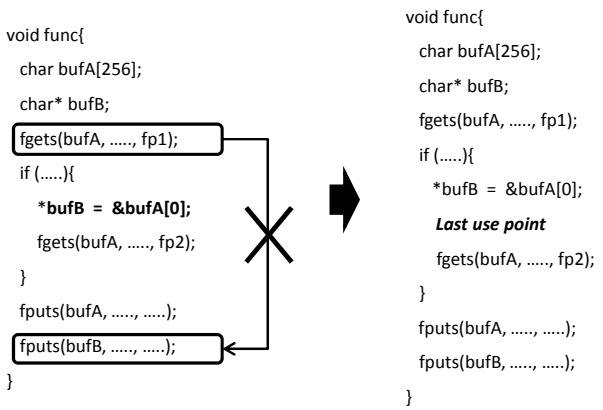

図 8 別名によるデータ伝播
Fig. 8 Data spread by alias.

用連鎖を結合できる．前項で述べた変数間の値の代入においては，定義-使用連鎖を結合することで，最終使用点の挿入時に問題が発生した．しかし，別名関係の変数では，参照元の変数が再定義された場合，**図 8** の左に示すように，別名の変数についての使用点には，定義点の値が伝播しない．つまり，結合された定義-使用連鎖は，参照元の変数のみを介したデータフローであると見なすことができる．そのため，変数の再定義が行われた場合，図 8 の右に示すように，その直前に最終使用点を挿入することで，使用点の共有により発生する問題を解決できる．

5. おわりに

本稿では，DF-Salvia のアクセス制御のためのデータフロー情報のプログラム解析について述べた．複数の実行経路を持つプログラムでは，定義-使用連鎖間において使用点の共有が行われる．そのため，DF-Salvia では，アクセス要求に対して適用する一意なポリシを決定できない．そこで，プログラム中に最終使用点を挿入することで，実行時にデータフローに適用されるポリシを 1 つに限定するための手法を提案した．本稿では，手続き内のプログラム解析のみを対象としたが，セキュアコンパイラの開発に向けて，手続き間，大局変数を含めたプログラム解析について検討することが今後の課題である．

参 考 文 献

1) 消費者庁: “個人情報の保護に関する法律,” http://www.cca.go.jp/seikatsu/kojin/houritsu/index.html
2) NPO日本ネットワークセキュリティ協会: “2008 年情報セキュリティインシデントに関する調査報告書 Ver.1.3,” http://www.jnsa.org/result/2008/surv/incident/2008incident_survey_v1.3.pdf,(2009).
3) 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用的データ保護方式,” 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).
4) Alfred V.Aho,Ravi Sethi,Jeffrey D.Ullman 著,原田 賢一 訳: “コンパイラ I/II 原理・技報・ツール,”サイエンス社,(1990).
5) Susan Horwitz,Alan J.Demers,Tim Teitebaum: “An efficient general iterative algorithm for dataflow analysis,” *Acta Informatica archive*,Vol.24,pp.679-694,(1987).
6) Jong-Deok Choi,Michael Burke,Paul Carini: “Efficient flow-sensitive interprocedural computation of pointer-induced aliases and side effects,” *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*,pp.232-245,(1993).