社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS2002-14 (2002-07)

プログラム依存グラフを利用した情報漏洩解析手法の提案と実装

西 秀雄† 横森 励士†† 井上 克郎†

† 大阪大学大学院情報科学研究科 〒560-8531 大阪府豊中市待兼山町 1-3
†† 大阪大学大学院基礎工学研究科 〒560-8531 大阪府豊中市待兼山町 1-3
E-mail: †{h-nisi,inoue}@ist.osaka-u.ac.jp, ††yokomori@ics.es.osaka-u.ac.jp

あらまし 情報漏洩解析とは, 重要な情報を扱うプログラムが情報の漏洩を引き起こさないかどうか確認するための手法である. プログラムの入力値に設定された機密度から出力値の機密度を求める情報漏洩解析が提案, 実現されている. 一方で, プログラム依存グラフと呼ばれるプログラム内の文間の依存関係を有向辺で表現したグラフが, プログラム構造解析の分野において利用されている. 既存の情報漏洩解析手法では, 解析時に繰り返し計算を行うため, 様々な機密度を入力値に与えて再計算を行うときに解析の時間的コストが問題となる. そこで本研究では, プログラム依存グラフを利用した情報漏洩解析手法の提案を行う. さらに, 束構造を持つセキュリティモデルに対する情報漏洩解析の実現方法を提案する. また, システムを試作し, 適用事例を通じてその有効性を検証する.

キーワード プログラム依存グラフ, セキュリティクラス, 情報フロー, 情報漏洩解析

A proposal and implementation of the information leak analysis using program dependence graph

Hideo NISHI†, Reishi YOKOMORI††, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1-3, Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan
†† Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan
E-mail: †{h-nisi,inoue}@ist.osaka-u.ac.jp, ††yokomori@ics.es.osaka-u.ac.jp

Abstract Information leak analysis is a mechanism which investigates information leakage from program data to the outside of the system, and the method which provides a secrecy level for each output of the program with respect to a given secrecy level for each input. On the other hand, a program dependence graph which represents dependency relations in the program is used in program slicing. The existing information leak analysis calculates by repetition calculation of the analysis in a procedure unit, so it takes many times to re-compute a program giving the various secrecy levels to input values. In this paper, we propose the realization method of the information leak analysis which uses program dependence graph with a lattice model. Moreover, we actually implement this system and confirme our approach.

Key words program dependence graph, security class, information flow, information leak analysis

1. まえがき

クレジットカード番号等, 第三者に知られてはならない情報を扱うプログラムや不特定多数の人間が利用するシステムにおける不適切な情報漏洩防止を目的として, Denning らはプログラム内のデータ授受関係を表す情報フロー (Information Flow) に基づき, 入力値が与えられたセキュリティに関するポリシーを保っているかどうか検証することで, 不適切な情報漏洩を検出する手法を提案した [2] [3]. また, 國信ら [10] は, 再起を含む手続き型プログラムに対応するため, プログラムに入力される値のセキュリティクラス (Security Class,SC) からプログラム中の各出力における SC を調査する手法を, 情報漏

洩解析（Information Leak Analysis）としてとして提案した．我々の研究グループは[10]の手法を[11]において実現した．

一方で，プログラム文間の依存関係を表すために，プログラム依存グラフ (Program Dependence Graph,PDG) と呼ばれる有向グラフが存在し，プログラムスライスの計算などに利用されている．PDG の各頂点はプログラム中の各文に対応し，各文間の依存関係はそれぞれの頂点を結ぶ有向辺で表される．

[11] では，[10] で提案されている手法を**データフロー方程式**（*Data Flow Equation*）[6] における繰り返し計算に基づいて行っている．この手法ではプログラム言語ごとにアルゴリズムを定める必要がある，複数回解析を行う場合の再利用性が低く，さらに，対応している SC の構造が 2 値のみであるといった問題点がある．

PDG の依存関係と情報フローは類似した性質を持っていることが知られている．そこで，本研究では PDG を探索することで情報漏洩解析を行う手法を提案する．この手法を用いる場合，PDG は言語独立であるため，この手法を用いることで他のプログラム言語への手法の移植も容易となる．また，一度 PDG を構築してしまえば，入力値を変更して再度解析を行う際も，同じ PDG を探索することで解析できる．そのため，再解析の時間的コストを抑えることが可能となる．

さらに，本研究ではこれまでの情報漏洩解析では実現されていなかった，束構造を持つセキュリティモデルに基づく情報漏洩解析手法を提案する．束構造を実現するために，束構造の任意の 2 元の最小上界を行列として表現し，演算時はその行列を逐次参照する方法を採用した．また，提案した実現手法に基づいて実際にシステムのプロトタイプを作成し有効性を検証する．評価では，前手法との実行時間の評価，例題に対する適用を行う．

以降 2. では既存の情報漏洩解析の手法について述べ，3. では PDG について，4. では新たな解析手法について述べる．5. ではシステムの実現を行い，6. で試作したシステムの検証を行う．

2. 情報漏洩解析

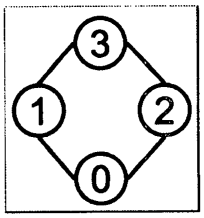
本節では，既存の情報漏洩解析について述べる．

2.1 セキュリティクラスと情報フロー

セキュリティクラス（*Security Class,SC*）とは各データの機密度を表す値で，データ d の SC を $SC(d)$ とする．同様に，**クリアランス**（*Clearance*）はユーザ（プロセス）がどの程度のデータまでアクセスできるかを表し，ユーザ u のクリアランスを $clear(u)$ とする．

システムにおける各 SC の機密度の強弱を表したものを**セキュリティモデル**（*Security Model*）といい，各元が SC と対応した束を用いて表される．セキュリティモデルでは任意の 2 元の最小上界，最小下界が定義されており，一意な最大元（*high*），最小元（*low*）が存在する．プログラム中の定数の SC は *low* である．セキュリティモデルの例を図 1-(a) に示す．

あるオブジェクト v からオブジェクト u へ，何らかの形で情報が推移していくとき，v から u への**情報フロー**（*Information Flow*）が存在すると定義する．情報フローにはその推移の仕方によって，**明示的フロー**（*Explicit Flow*），**暗示的フロー**（*Implicit Flow*）の 2 種類が存在する．プログラム中の文 s_1 で変数 v を定義しており，文 s_2 でその変数 v を参照して変数 u が定義されているとき，u の値は v によって決まるため，v から u に明示的フローが存在すると定義する．また文 s_1 が変数 v を参照する条件文であり，その条件文のブロック内に変数 u を定義する文 s_2 が存在するとき，v の値によって文 s_2 が実行されるかどうかが決まる，そのため v から u に暗示的フローが存在すると定義する．



$low = 0, high = 3$

(a) セキュリティモデル

```
{各変数のSCの初期値
 SC(a)=1, SC(b)=0, SC(c)=2, SC(d)=0 }

 1: b := a + 1;
      { SC(b)=SC(a)∨0 }
 2: if (c > 0) then begin
 3:    d := a + 1
      { SC(d)=SC(c)∨SC(a)∨0 }
 4: end;

{終了後の各変数のSC
 SC(a)=1, SC(b)=1, SC(c)=2, SC(d)=3 }
```

(b) 情報漏洩解析の例

図 1　セキュリティモデルと情報漏洩解析の例

図 1-(a) のようなセキュリティモデルを持つプログラム上で，$SC(x) = 1$，$SC(y) = 2$, $SC(z) = 0$ のとき，x と y の和を z に代入するときを考える（z := x + y）．このとき x と y の値が z にフローし，z は x と y 両方のデータに関する情報を持つことになる．このとき z を参照できるのは，x と y 両方のデータを読むことができるクリアランスを持つユーザであるべきである．よって，代入後の $SC(z)$ は，$SC(x)$ と $SC(y)$ の最小上界となり $SC(z) = 3$ となる．このように，複数のデータからの情報フローを受ける変数は，入力となるデータの最小上界である SC を持つことになる．

図 1-(b) に，図 1-(a) のようなセキュリティモデルを持つプログラム上での，情報漏洩解析を示す．$SC(x)$ と $SC(y)$ の最小上界を取る演算を SC の和演算といい $SC(x) \vee SC(y)$ と示す．

2.2 既存の情報漏洩解析手法

情報漏洩を防ぐアクセス制御法として，*Mandatory Access Control* [4] と呼ばれる制御法が存在する．この制御法のもとでは，$clear(u) \geq SC(d)$ のときかつそのときのみ，ユーザ u はデータ d を読むことができる．

しかし，このアクセス制御法のもとでは，クリアランスが $SC(d)$ 以上のユーザプログラムがデータ d を読み込み，それを，故意にまたは過失によって，クリアランスが $SC(d)$ より小さいユーザにもアクセスできる記憶域に書き出してしまうと，不適切な情報漏洩が生じる．そのような不適切な情報漏洩を防ぐために，**情報漏洩解析**（*Information Leak Analysis*）と呼ばれる手法が提案されている．

情報漏洩解析は，まず Denning らによって入力値の SC と出力域のクリアランスとの矛盾を検出する手法が提案された [2] [3]．また [7] では，[3] の手法を理論的に再検討し解析手法の一般化を試みている．しかし，これらの手法では再帰手続きや大域変

数を考慮していないこともあり，解析対象となるプログラムが単純な構造のものに限られていた．また，関数呼び出し文に対する解析の際，戻り値の判定は実引数全体に対してのみで，戻り値の計算に実際にどの引数の値が利用されているかを考慮していないなど不正確な面があり，実際に関数内部を解析すれば矛盾がないことがわかる場合でも，矛盾を検出してしまうなど，解析が厳密になりすぎてしまうという欠点があった．

そこで，国信らは，再帰を含む手続き型プログラムに対する情報フロー解析アルゴリズムを提案した[10]．この方法では，解析対象プログラム中すべての実行文について，その文の実行前後の各変数のSC間で成り立つべき再帰的な関係式を定義する．この関係式に基づき，プログラムの各手続きの実行結果のSCを解析する．プログラムのmain関数の仮引数が $x_1, \ldots, x_i$，入力ファイルが $infile_1, \ldots, infile_j$ であるとき，実引数と入力ファイルに対応するi + j個のSCの組が与えられると，それらを元に上記の関係式を同時に満たす最小解が繰り返し計算により求められる．そして，main関数の戻り値が y_1，出力ファイルが $outfile_1, \ldots, outfile_k$ であるとき，この解が戻り値と出力ファイルに対応する1 + k個のSCとなる．我々の研究グループでは提案されている手法を[11]において実現した．

3. プログラム依存グラフ

プログラム依存グラフ(*Program Dependence Graph,PDG*)はプログラム内の文の依存関係を表す有向グラフである[1]．PDGの頂点はプログラム中の各文，及びif文やwhile文の条件判定部分を表す．2種類の辺が存在し，それぞれ変数の影響を表す**データ依存辺**，および条件文や繰り返し文の制御の影響を表す**制御依存辺**が存在する．

PDGの構築は次のような手順をたどる．

Phase 1:　依存関係解析 (データ依存関係，制御依存関係)

Phase 2:　PDGの構築

Phase1では，プログラム中の文間の依存関係解析を行う．依存関係には以下の制御依存関係とデータ依存関係の2種類がある．

- データ依存関係

以下の3つの条件を全て満たすとき，文 s_1 から文 s_2 へ変数 v に関して**データ依存** *(Data Dependence, DD)* があると定義し，s_1 に対応する頂点から s_2 を表す頂点に制御依存辺を引く．

- 文 s_1 で変数 v を定義している．
- 文 s_2 で変数 v を参照している．
- 文 s_1 から文 s_2 への実行可能なパスのうちで，s_1 から s_2 間に変数 v を再定義している文が存在しないパスが存在する．

- 制御依存関係

以下の条件を全て満たしているとき，文 s_1 から文 s_2 への**制御依存** *(Control Dependence, CD)* があると定義し，s_1 に対応する頂点から s_2 を表す頂点に制御依存辺を引く．

- 文 s_1 が条件文である．
- 文 s_2 が実行されるかどうかは，文 s_1 の結果に依存する．

Phase2では，Phase1での依存関係解析で得られた依存関係を元に，各頂点をデータ依存辺，制御依存辺で結ぶことで，PDGを構築する[8]．

4. PDGを用いた情報漏洩解析

本節では，2.，3.の内容をふまえ，既存の情報漏洩解析手法の問題点と，本研究で提案する新たな手法について述べる．

4.1 既存の情報漏洩解析手法の問題点

[11]の手法は，各文内・文間における情報フローに基づき，データフロー方程式に基づいて，文実行前後において各変数のSC間で成り立つべき再帰的な関係式を定義し，その関係式を同時に満たす最小解を繰り返し計算によって求めることで，出力値のSCを得る手法である．

PDGによって表されるプログラム内部の依存関係は，情報フローの場合と同じようにデータ依存・制御依存の2種類に分類でき，それぞれの性質も情報フローの場合と非常に類似している．そこで，データ依存辺と明示的フロー，制御依存辺と暗示的フローを対応させ，アルゴリズム中のデータ依存辺を構築する部分を，明示的フローの計算部に，制御依存辺を構築する部分を暗示的フローの計算部に置き換え，解析を行なっている．

しかし，[11]の手法は，データフロー方程式における繰り返し計算に基づいており，解析アルゴリズムを言語ごとに定める必要があるため手法としての汎用性に乏しい．また，プログラムの実行順に従い繰り返し計算を行っているため，階層化しその中で判定される変数の数が多いプログラムに対しては，解析の繰り返しの条件となる変数が多くなることで繰り返し回数が増加し，効率が落ちる．このため，入力文のSCの初期値を変更するたびに再度繰り返し計算が必要となり，繰り返し解析を行う場合に効率が悪い．さらに，SCを $high, low$ の2値のみしか実現していないため，複雑な束構造を持つSCを対象とした情報漏洩解析を行うことができず実用性に乏しい，といった問題点がある．

4.2 新たな情報漏洩解析手法の提案

我々は既存の手法の問題点をふまえて，PDGを探索することにより情報漏洩解析を行う手法を提案する．これにより，PDGの言語独立性を利用することによる言語独立な情報漏洩解析が可能となる．再解析における時間的コストの減少が可能となる．また，束構造を持つセキュリティモデルに対する情報漏洩解析を行うために，ユーザがあらかじめセキュリティモデルの束を記述し，それを参照してSCの和演算を行う仕組みを実現した．

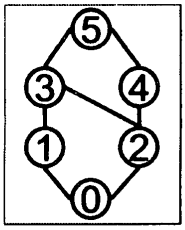
4.2.1 情報フローとPDGにおける依存関係

2.で述べた情報フローと，3.で述べたPDGにおける依存関係は，共にプログラム内部のデータがどのように推移していくかを示しており，非常に似た性質を持っている．情報フローにおける明示的フロー，暗示的フローは，それぞれPDGの依存関係におけるデータ依存関係と，制御依存関係に対応しているため，PDGを探索することで情報フローの解析を行うことができる．

4.2.2 束構造で表されるセキュリティモデルの実現

束の任意の二元の最小上界を二次元行列を用いて表すことで，束構造を持つセキュリティモデルへの情報漏洩解析に対応する．

図2-(a)のようなハッセ図で表される束構造を考える．low



$low = 0, high = 5$

(a) ハッセ図

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	1	3	3	5	5
2	2	3	2	3	4	5
3	3	3	3	3	5	5
4	4	5	4	5	4	5
5	5	5	5	5	5	5

(b) 束構造の和演算を表す二次元行列

図 2　束構造の表現

は 0，$high$ は 5 と対応している．このとき，和演算結果を格納した二次元行列を用いることで，図 2-(a) のハッセ図で表される束構造は，図 2-(b) の二次元行列として表現できる．二次元行列の m 行 n 列の要素が $SC(m) \vee SC(n)$ の結果を表す．

また $SC(a) \vee SC(b)$ の結果が $SC(c)$ であるとき，

- $SC(a) = SC(b) = SC(c)$ ならば $SC(a) = SC(b)$
- $SC(a) = SC(c)$, $SC(b) \neq SC(c)$ ならば $SC(a) > SC(b)$
- $SC(b) = SC(c)$, $SC(a) \neq SC(c)$ ならば $SC(b) > SC(a)$
- $SC(a) \neq SC(c)$, $SC(b) \neq SC(c)$ ならば $SC(a)$, $SC(b)$ 間に大小関係なし

というような関係が成り立っているため，大小関係の比較も二次元行列を用いて行うことができる．

4.3　PDG を利用した束構造をもつセキュリティモデルに対する情報漏洩解析

ここでは PDG を利用した，束構造のセキュリティモデルに対する情報漏洩解析の手順について説明する．初期状態として，全ての頂点における SC の初期値は low とする．解析手順は次の通りである．

Phase 1:　依存関係解析 (データ依存関係，制御依存関係)
Phase 2:　PDG の構築
Phase 3:　前提条件の入力
Phase 4:　入力文を起点とした PDG の探索

ここでは Phase3，Phase4 について説明する．

Phase 3: 前提条件の入力

解析対象となるプログラムの 2 つの前提条件を入力する．

- セキュリティモデル:
  二次元行列の形でセキュリティモデルの定義を入力する．
- 各入力文で読み込まれる値の SC:
  各入力文で読み込まれる値の SC を入力する．

Phase 4: 入力文を起点とした PDG の探索

PDG の探索をしながら，経路上の各頂点で SC の和演算を行う．

解析対象のプログラム中の各入力文を起点として，PDG の探索を行う．探索は，頂点から出るデータ依存辺および制御依存辺を順方向にたどる．変数 v の値が読み込まれる入力文を起点とした場合，探索で到着した頂点において，その頂点の現在の SC と $SC(v)$ の和演算を行い，頂点を演算結果の SC で更新する．その頂点が和演算を実行して更新されない場合，その頂点からの辺はたどらない．一方，更新される場合，その頂点からの辺をたどり次の頂点へ移る．SC が更新される頂点がなくなった場合，次の入力文を起点として PDG 探索を行い，全ての入力文に対して PDG の探索を行う．

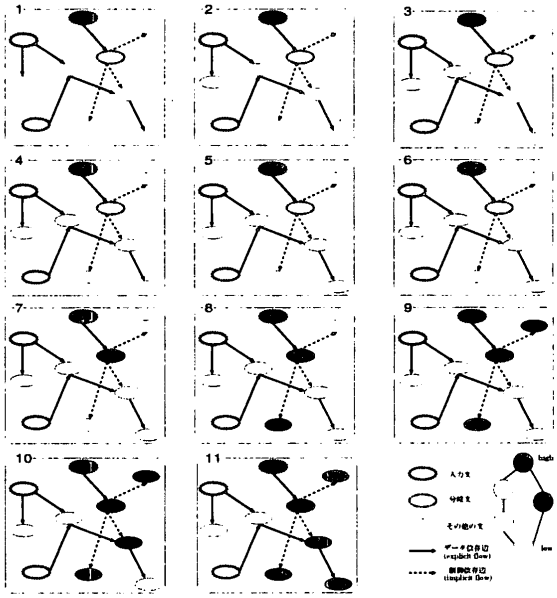


図 3　解析の例

4.4　解析の例

提案した手法により情報漏洩解析を行なった時の解析例を図 3 に示す．図 3-1 が前提条件を入力した直後の PDG である．SC の束構造は図の右下に示す．入力文を表す頂点にセットされた SC は頂点の色の濃さで示す．起点とした入力文に対応する頂点からの SC が辺を順方向にたどって PDG の各頂点の SC を更新していく．図 3-11 が最終的な結果となる．

5.　PDG を用いた情報漏洩解析システム

本節では，本研究で実現した情報漏洩解析ツールの概要について述べる．

5.1　実装の方針

本ツールの実現は，我々が開発したスライスツールである *Osaka Slicing System, OSS* [5], [9] に情報漏洩解析部を機能追加する形で行った．入力ファイル，出力ファイルはそれぞれ標準入力，標準出力に限定している．また，次に述べる SC 制約機能を追加した．

5.2　SC 制約機能

情報漏洩解析の問題点として，SC の高いデータに対してプログラム中で暗号化などを行って，もとのデータが隠蔽された場合でも，SC は高いままとなってしまうことが考えられる．実際に運用する際には，暗号化などが信頼できるとユーザが判断したときには，プログラム中の関数による出力の SC を，ユーザが任意に設定できる機能が必要である．そこで今回の実装では，関数の戻り値の SC をユーザが任意に設定できる機構を実現した．Phase 3: 前提条件の設定時の際，関数の戻り値の SC

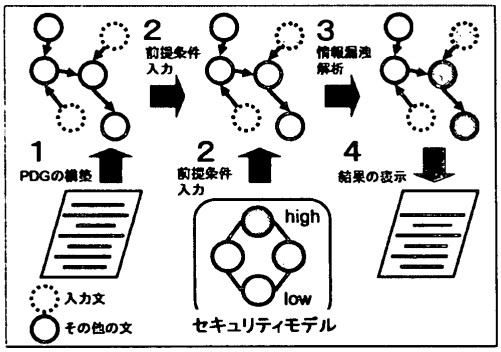


図 4　解析の流れ

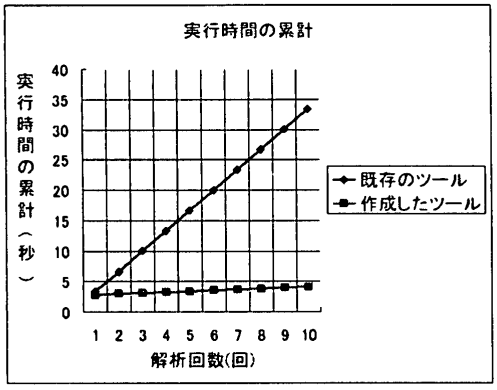


図 5　実行時間の測定結果
(実験環境:Pentium4 2GHz，メモリ 1GB，Free-BSD4.5)

を設定することで，その関数の戻り値を表す頂点から辺をたどる際に，その頂点で設定した SC をもとに，PDG を探索することができる．この機能により，SC の高いデータが関数内で隠蔽され，現実的にはその隠蔽されたデータからもとのデータを類推することが不可能であるとき，その値の SC を強制的に低くすることで，現実的な安全度に則した情報漏洩解析を行なうことができる．

5.3　ツールの解析の流れ

ツールの解析の流れを図 4 に示す．構文・意味解析部は，UI 部からの要求に応じて構文解析，意味解析を行なう（図 4-1）．次に，ユーザはソースファイル上で情報漏洩解析の前提条件を設定（図 4-2）し，UI 部を通じて情報漏洩解析部へ依頼する．情報漏洩解析部は，前提条件を基に解析を行ない（図 4-3）その結果を UI 部に渡す．UI 部は，各々の文の SC に応じて背景色を変更することで SC を表示する（図 4-4）．

6.　検　　証

本節では 5. 節で紹介した情報漏洩解析システムの具体的な適用事例を取り上げその有効性を検証する．検証内容は，同条件で情報漏洩解析を行ったときの既存のシステムとの実行時間の比較．複雑な束構造を SC として持つサンプルプログラムに対するシステムの適用である．

6.1　前手法との実行時間の比較

既存のシステムと作成したシステムについて同条件で解析を繰り返し行ない，その実行時間の累計を比較した．図 5 に結果を示す．図の通り，PDG 構築を行う必要がある一度目の解析

ユーザ	ユーザが参照できるデータ
学生	自身の一般教養科目分野の成績 自身の専門科目分野の成績 自身の認証番号
一般教養科目分野の事務員	全ての学生の一般教養科目分野の成績 一般教養科目分野の事務員の認証番号
専門科目分野の事務員	全ての学生の専門科目分野の成績 専門科目分野の事務員の認証番号

表 1　各ユーザが参照できるデータ

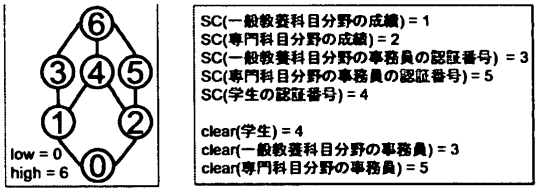


図 6　システムのセキュリティモデル

においては実行時間にほとんど違いはないが，解析を繰り返し，回数を重ねるに従って既存のシステムと作成したシステムの実行時間の累計に開きが出てくる．既存のシステムは一度目の解析にかかる時間と以降の解析の時間が変わらない．しかし，作成したシステムでは二度目からの解析時間は減少している．これは作成したシステムが一度構築した PDG の持つ情報を繰り返し利用できるためである．

6.2　学生の成績管理システムに対する適用事例

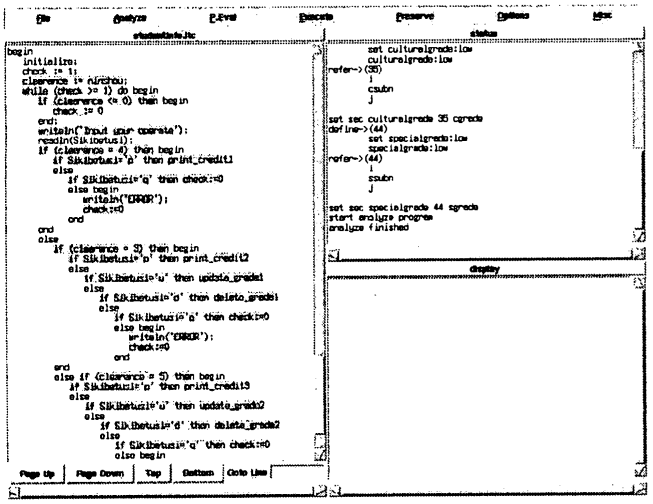
次に複雑なセキュリティモデルを持つプログラムの安全性の確認に本システムを適用する．プログラムの安全性の確認とは，プログラムを静的に解析し SC の高い出力を事前に検出することで，予想外の情報漏洩を防ぐことである．．プログラムの安全性を確認し対策を立てることで，SC の高い出力文を減らし，情報漏洩の可能性をより低くすることができる．

適用対象として学生の成績管理システムを考えた．成績は一般教養科目分野と，専門科目分野に分類され，それぞれ「可」または「不可」のいずれかとなる．全ての学生は自身の成績を，分野に限らず参照できる．また，一般教養科目分野，専門科目分野にはそれぞれ事務員が居り，一般教養科目分野の事務員は全ての学生の一般教養科目の成績の参照と書き換えを行うことができるが，専門科目分野の成績を参照，書き換えすることはできない．同様に，専門科目分野の事務員は，全ての学生の専門科目分野の成績の参照と書き換えを行うことができるが，一般教養科目分野の成績を参照，書き換えすることはできない．

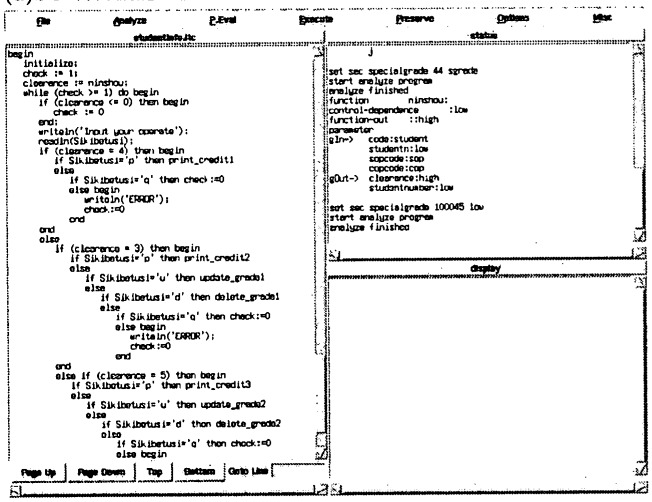
学生，一般教養科目分野の事務員，専門科目分野の事務員はそれぞれ認証番号をもっており，成績管理システムにアクセスする際にそれぞれの認証番号を入力する．入力された認証番号によりシステムはユーザが何者かを判断し，それぞれのユーザに応じた権限を与える．そのユーザは以後その権限に応じた操作を行うことができる．

各ユーザの参照できるデータを図 1 に整理する．

このシステムにおけるセキュリティモデルは図 6 のハッセ図のようになる．図 6 をもとに，それぞれのデータの SC と，ユーザのクリアランスの SC に対するマッピングを考える．主要な



(a)SC 制約機能の使用前



(b)SC 制約機能の使用後

図 7 解析結果

データとその SC は,

- SC(一般教養科目分野の成績) = 1
- SC(専門科目分野の成績) = 2
- SC(一般教養科目分野の事務員の認証番号) = 3
- SC(専門科目分野の事務員の認証番号) = 5
- SC(学生の認証番号) = 4

同様にユーザとそのクリアランスは,

- $clear$(学生) = 4
- $clear$(一般教養科目分野の事務員) = 3
- $clear$(専門科目分野の事務員) = 5

となる.

サンプルプログラムの各々のデータに対して上で述べた SC を与えて解析を行なった結果, 33 個の出力文のうち, SC が 4 の出力文が 1 個, 5 の出力文が 1 個, 6 の出力文が 26 個という結果が得られた. (図 7-(a))

これは, 各ユーザの認証番号が様々な暗示的フローを引き起こし, そのためユーザの認証を行なう関数が現在のユーザのクリアランスを判別した結果として返す値の SC が高くなり, その値が引き起こす暗示的フローを受ける文が多数あったために起こった. しかし, 各ユーザの認証番号や, 関数が返す値が十分に隠蔽されたときを想定して, 認証関数の戻り値の SC を *low* に設定して再度解析を行なうと, (図 7-(b)) 33 個の出力文のうち, SC が 0 の出力文が 27 個, 1 の出力文が 2 個, 2 の出力文が 2 個, 4 の出力文が 1 個, 5 の出力文が 1 個という結果が得られた. このことから, 適切なデータに隠蔽を行なえば, 現実的な情報漏洩の可能性が大きく下がることがわかる. このように, 情報漏洩解析と SC 制約機能を組み合わせることで, どのデータに隠蔽を施すべきかを判断する指針にすることができる.

7. まとめと今後の課題

本研究では, PDG を利用した, 束構造を持つセキュリティモデルに対する情報漏洩解析を行なう手法を提案し, 実現した. また, より現実的な利用を目指し, 関数の戻り値をユーザが任意に設定できる機構を実装した. さらに, 適用例を通じて, その有効性を検証した. 本手法では, プログラムスライスなどに利用される, 依存関係を示した PDG を使用することで, 再利用性向上による解析コストの低下と, 言語に依存しない情報漏洩解析を行うことができた. また, 行列を用いて演算を表現することで, 束構造を持つセキュリティモデルに対する情報漏洩解析に対応している.

今後の課題として, オブジェクト指向言語に対する情報漏洩解析アルゴリズムの拡張と実装が挙げられる.

文 献

[1] T. Reps, S. Horwitz, M. Sagiv and G. Rosay, "Speeding up Slicing", In Proceedings of the ACM SIGSOFT '94 Symposium on the Foundations of Software Engineering, pp. 11–20, 1994.

[2] D.E.Denning. "A Lattice Model of Secure Information Flow", Communication of the ACM,Vol. 19, No. 5, pp. 236–243, 1976.

[3] D. E. Denning and P. J. Denning, "Certification of Programs for Secure Information Flow", Communication of the ACM, Vol. 20, No. 7, pp. 504–413, 1977.

[4] G. Purnul, "Database Security", Advances in Computers(M.Yovits Ed.),Vol. 38, pp. 1–72, 1994.

[5] Yoshiyuki Ashida, Fumiaki Ohata and Katsuro Inoue, "Slicing Methods Using Static and Dynamic Information", Proceedings of the 6th Asia Pacific Software Engineering Conference (APSEC'99), Takamatsu, Japan, pp. 344–350, 1999.

[6] Alfred V. Aho, Ravi Sethi and Jeffrey D. Ullman, "Compilers : Principles, Techniques, and Tools", Addison-Weseley, 1986.

[7] J.Banâtre, C.Bryce and D.Le Mêtayer, "Compile time Detection of Information Flow in Sequential Programs", Proc.3rd ESORICS, LNCS 875, pp. 55–73, 1994.

[8] 植田, 綿, 井上, 鳥居: "再帰を含むプログラムのスライス計算法", 電子情報通信学会論文誌, Vol. J78-D-I, No. 1, pp. 11–22, 1995.

[9] 佐藤, 飯田, 井上, "プログラムの依存関係解析に基づくデバッグ支援システムの試作", 情報処理学会論文誌, Vol. 37, No. 4, pp. 536–545, 1996.

[10] 國信, 高田, 関, 井上, "束構造のセキュリティモデルに基づくプログラムの情報フロー解析", 電子情報通信学会技術研究報告,2000 年 11 月.

[11] 横森, 大畑, 高田, 関, 井上, "セキュリティ解析アルゴリズムの実現とオブジェクト指向言語への適用に関する一考察", 電子情報通信学会ソフトウェアサイエンス研究会,2000 年 11 月.

〔盛 文 社〕