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.

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
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, 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, 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いう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存在する．既存の情報漏洩解析手法では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, そ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，一般的な束構造を持つセキュリティクラス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 NISI†, Reisi YOKOMORI†, and Katuro INOUE†

† Graduate School of Information Science and Technology, 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 technique for checking whether the program does not cause an informational disclosure, and information leak analysis which computes the degree of secrecy of output values from the degree of secrecy of input value is proposed and realized. On the other hand, there is a program dependence graph which is a digraph representing dependence relations among program's sentences. The existing information leak analysis is based on repetition calculation, so it takes many time to recompute giving the various degrees of secrecy to input values. In this research, propose the realization method of the information leak analysis to a security class with general lattice. Moreover, a system is actually made as an experiment and the validity is verified through an application example.
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 らはプログラムを静的に解析し，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 [3].

より現実的な手法として，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 を調査する手法が，**情報漏洩解析**（*Information Leak Analysis*）として國信ら [15] によって提案され，我々の研究グループでは提案されている手法を [16] によって実現した．

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する．PDG の各頂点はプログラム中の各文および条件判定部分に対応し, 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. PDG の利点として, 再利用性があげられ, プログラムスライスの計算などに利用されている.

[16] では, [15] で提案されている手法をデータフロー方程式 [8] における繰り返し計算に基づいて行っており，様々な機

密度を入力値に与えて再解析を行うとき時間的なコストがかかる，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，などの問題点がある．また，対応している SC が 2 値のみであるといった問題点もある．

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, 一度 PDG を構築してしまえば，入力値の機密度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．また, PDG は言語独立であるため，他のプログラム言語への手法の移植も容易となる．

さらに，これまでの情報漏洩解析では実現されていなかった，一般的な束構造をもつ SC に基づく情報漏洩解析手法を提案する．一般的な束構造を実現する方法として，束構造上での和演算を行列として表現し，あらかじめ保存する．演算の際にはその行列を逐次参照することで，束構造上での和演算を行うことができる．

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

情報漏洩を防ぐ手段として，*Mandatory Access Control* と呼ばれる次のようなアクセス制御法がよく用いられる [4]:

> 「各データに対してその機密度を表す**セキュリティクラス**（*Security Class,SC*）を割り当てる．データ d の SC を $SC(d)$ と表す．同様に，ユーザ（プロセス）に対し，どの程度のデータまでアクセスできるかを表す**クリアランス**（*Clearance*）を割り当てる．ユーザ u のクリアランスを $clear(u)$ と表す．$clear(u) \geqq SC(d)$ 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.」

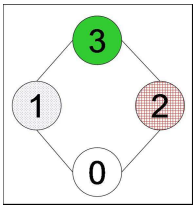
しかし，このアクセス制御法の場合，クリアランスが $SC(d)$ 以上のユーザプログラムがデータ d を読み込み，それを，故意にまたは過失によって，クリアランスが $SC(d)$ より小さいユーザにもアクセスできる記憶域に書き出してしまうと，不適切な情報漏洩が生じる．そのような不適切な情報漏洩を防ぐための手法を，情報漏洩解析という．

情報漏洩解析には，目的に応じて

- プログラムの入力値の SC から出力値の SC を求める [15]
- 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 [3]

の 2 つの手法があるが，本稿では前者に着目している．

本節では, [15] で提案され, [16] で実現された，既存の情報漏洩解析について述べる.



$low = 0, high = 3$

図 1　セキュリティモデル

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*）とはデータの機密度を表す値である．システムにおける各 SC の機密度の大小関係を表したものを**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．セキュリティモデルの例を図 3 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*）が存在するという．情報フローにはその情報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*）という 2 種類が存在する．ソースプログラム中の文 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，変数 v の値が変数 u にフローするので，変数 v から変数 u に明示的フローが存在する．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，変数 v の値によって文 s_2 が実行されるかどうかが決まる，すなわち変数 u の値が変数 v の値によって決められるので，変数 v から変数 u に暗示的フローが存在する.

図 3 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 にフローする．フローにより，z には 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．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．プログラム中に現れる定数の SC は low である．$SC(x)$ と $SC(y)$ の最小上界を取る演算を SC の和演算といい $SC(x) \bigvee SC(y)$ と示す.

図 2 に，図 3 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.

2.2 既存の情報漏洩解析手法

プログラムの入力値の SC から出力値の SC を求める手法が, 國信ら [15] によって提案され，我々の研究グループでは提案されている手法を [16] によって実現した. [16] において計算に利用される手法は，各文内・文間における情報フローに基づき，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

```
{ 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:
  3:  if (c > 0) then begin
  4:    d := a + 1
        { SC(d)=SC(c) ⋁ SC(a) ⋁ 0 }
  5:  end;

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

図 2　情報漏洩解析の例

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．

解析中に参照される変数が持つ SC を表現する集合として，要素が (変数, SC) の組である**セキュリティクラス集合**（*Security Class Set*，以降，SCset と省略する）を定義する．[16] では，簡単のために，SC は $high, low$ に限定しており，全ての変数の SC の初期値は low である．解析では，プログラムの実行順に従い SCset を逐次更新することで出力文における SC を求める．

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 [2]．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し，辺は変数の影響を伝える**データ依存関係**，及び条件文や繰り返し文の制御の影響を伝える**制御依存関係**を表す．

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，まずソースコードを入力とし，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．

- 制御依存関係

今，ソースプログラム p 中の文 s_1，s_2 について考える．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)* があるという．

- 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．

- データ依存関係

今，ソースプログラム p 中の文 s_1，s_2 について考える．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)* があるという．

- 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で，そのパスにおいて文 s_1 から文 s_2 間に変数 v を再定義している文が存在しない，というものが存在する．

Phase2 では，Phase1 での依存関係解析で得られた依存関係情報を元に PDG を構築する [13]．

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

2. で述べた既存の情報漏洩解析手法は繰り返し計算に基づいているため，手続きごとに繰り返し計算が必要となる．解析の条件となる各変数の SC が変更されるたびに解析を行うので，特に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．さらに，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る．このため，大規模なプログラムにおいて，様々な SC の初期値を入力文に与えて再解析を行う場合に時間的コストが問題となる．また，解析アルゴリズムを言語ごとに定める必要があるため汎用性に乏しい．

その他，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．

4.2 新たな情報漏洩解析手法の提案

既存の手法では 4.1 で述べたような問題点が考えられる．そこで我々は新たな情報漏洩解析の手法として，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．これにより再解析における時間的コストの減少と，PDG 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．また，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 の和演算を行うことで，一般的なセキュリティモデルに対する情報漏洩解析を実現した．

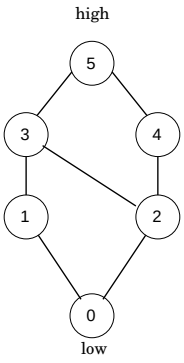
4.2.1 情報フローと PDG における依存関係

2. で述べた情報フローと，3. で述べた PDG における依存関係を考える．定義より，情報フローと 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つ．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．すなわち PDG におけるデータ依存辺と制御依存辺は，明示的フローと暗示的フローをそれぞれ表しているので，ある入力文において定義されている変数についての情報フローは，その入力文に対応する頂点を起点として PDG を探索することによって得られる．

4.2.2 一般的な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，一般的なセキュリティモデルへの情報漏洩解析に対応する．

図 3 のように，束の任意の二元の最小上界を二次元行列で表す．例えば，左のハッセ図で表される束構造は，右の二次元行



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

束構造の和演算を表す二次元行列

図 3　束構造の表現

列として表される．low は 0，$high$ は 5 と対応している．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 $SC(a) \vee SC(b) = SC(c)$ のとき，

- $SC(a) = SC(b) = SC(c)$ 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

以上が成り立ち，二次元行列から大小関係も計算できる．

4.3　PDG を利用した一般的な束構造に対する情報漏洩解析

全ての頂点において 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:　PDG の探索と SC の和演算
Phase1，Phase2 はすでに 3. で示した．

前提条件の入力

Phase3 では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:
4. 2. 2 で示した形式で表された定義を入力．
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．

PDG の探索と SC の和演算

Phase4 では 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．

PDG の探索は，解析対象のプログラム中の各入力文を起点とする．起点とした入力文で変数 v の値が読み込まれるとする．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．次の辺をたどる前に，次の頂点が和演算を実行して更新されないならばその辺はたどらない．更新されるならば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れば次の入力文を起点として PDG 探索と SC 和演算を行う．

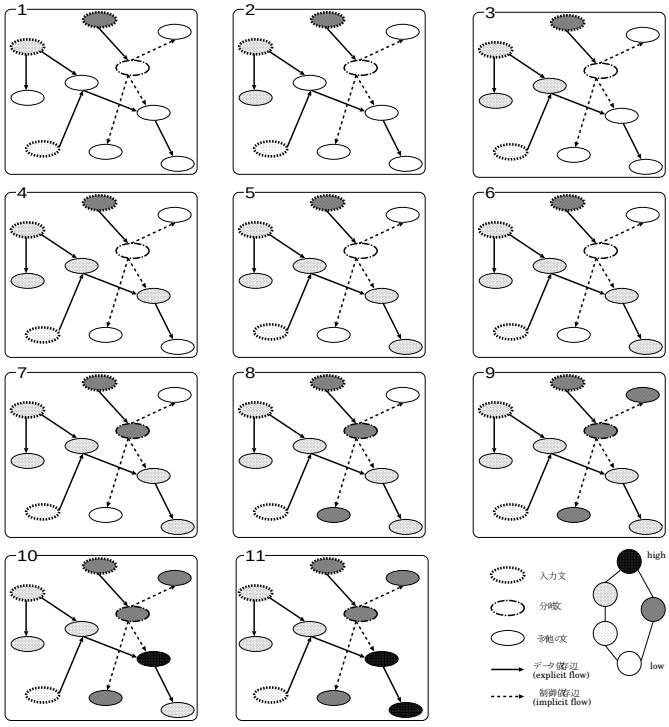


図 4　解析の例

4.4　解析の例

4. 3 で示したアルゴリズムにより情報漏洩解析を行なった時の様子を図 4 に示す．図 4-1 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．起点とした入力文の SC が辺をたどって PDG の各頂点の SC を更新していく．もし次の頂点の SC が，起点とした入力文の SC より大きければその辺はたどらない．たどる辺が無くなれば，次の入力文を起点として再び探索を行なう．図 4-11 が最終的な結果である．

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* [6], [14] に情報漏洩解析部を機能追加する形で行った．

対象言語は Pascal のサブセットである．また，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

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やマージを行って，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．

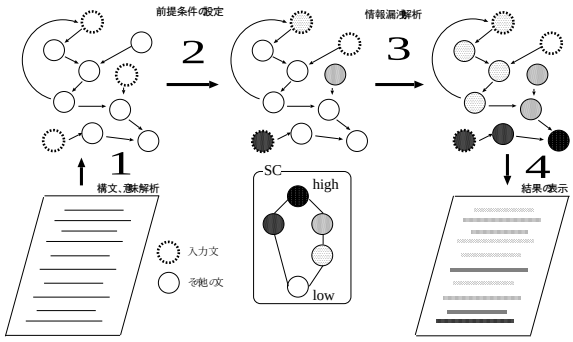


図 5　解析の流れ

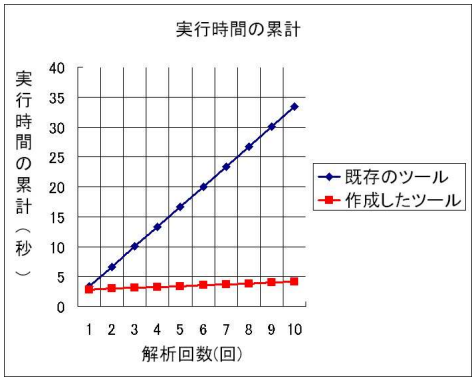


図 6　実行時間の測定結果
(実験環境:Pentium4 2GHz，メモリ 1GB，Free-BSD4.5)

5.2.1　実 現 方 法

今回の実装においてはその機能として，関数の戻り値の SC をユーザが任意に設定できるように実現した．手順としては前提条件の設定時の際，関数の戻り値を設定できる．具体的には，PDG 中のその関数の exit 頂点の SC に探索の際に与えられた SC をセットし，さらに以後の探索において，その頂点から情報フローするデータの SC をセットされた SC とする．以降はセットした SC を持つデータが情報フローすることになる．

この機能により，例えばプログラム中で関数の戻り値に与えられる 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 5 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 5-1）．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 5-2）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 5-3）その結果を UI 部に渡す．UI 部は，ユーザが定義した各 SC の色を各々の文の背景色とすることで，各々の文の SC を表示する（図 5-4）．

6.　検　　証

本節では 5. で作成した情報漏洩解析アルゴリズムを実現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．6.1 で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違いについて述べる．また，6.2 では複雑な束構造を SC として持つサンプルプログラムを考え，それに対してシステムの適用を行った．

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する．図 6 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．

このことからプログラムが大規模になった場合，本手法は効率的である．

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をいう．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．

学生の成績管理システムを考える．成績は一般教養科目分野と，専門科目分野に分類でき，それぞれ「可」または「不可」のいずれかである．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．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．

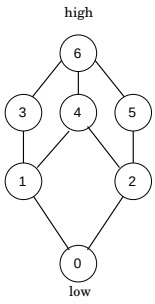
各ユーザの参照できるデータを整理すると，

学生:　自身の一般教養，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．

となる．



ハッセ図

図 7　システムのセキュリティモデル

以上より，データとユーザから考えた，このシステムのセキュリティモデルを図 7 のハッセ図に示す．図 7 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．主要なデータとその SC は，データ d の SC を $SC(d)$ と表すと，

一般教養科目分野の成績 $cgrade$:　$SC(cgrade) = 1$

専門科目分野の成績 $sgrade$:　$SC(sgrade) = 2$

一般教養科目分野の事務員の認証番号 $copcode$:　$SC(copcode) = 3$

専門科目分野の事務員の認証番号 $sopcode$:　$SC(sopcode) = 5$

学生の認証番号 $studentcode$:　$SC(studentcode) = 4$

同様にユーザとそのクリアランスは，ユーザ u のクリアランスを $clear(u)$ と表すと，

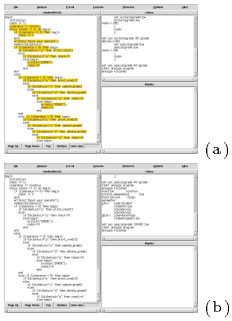
学生 $student$:　$clear(student) = 4$

一般教養科目分野の事務員 cop:　$clear(cop) = 3$

専門科目分野の事務員 sop:　$clear(sop) = 5$

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 8-(a))



(a)

(b)

図 8　解 析 結 果

これは，各ユーザの認証番号が様々な implicit flow 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 implicit flow 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 8-(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，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，5.2 で述べた機能は，どのデータに隠蔽を施すべきかを判断する指針にもなる．

7.　まとめと今後の課題

本研究では，PDG を利用した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能を実装した．複雑なセキュリティモデルを持つデータベースシステムを試作し，実現したツールをそのプログラムに対して適用し，その有効性を確かめた．

今後の課題として，

- 情報フローの定義の拡張
- 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

などが挙げられる．

文　　献

[1] M. Weiser, “Program Slicing”, In Proceedings of the 5th International Conference on Software Engineering, pp. 439–449, 1981.

[2] 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.

[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] S. Horwitz. and T. Reps. , “The Use of Program Dependence Graphs in Software Engineering”, Proceedings of the 14th International Conference on Software Engineering, pp. 392–411, 1992.

[6] 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.

[7] D.C. Atkinson and W.G. Griswold, “The design of whole-program analysis tools”, Proceedings of the 18th International Conference on Software Engineering, Berlin, Germany, pp. 16–27, 1996.

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

[9] Harman, M. and S. Danicic, “Using program slicing to simplify testing”, Journal of Software Testing, Verification and Reliability, Vol. 5, No. 3, pp. 143–162, 1995.

[10] Binkley, D., Horwitz, S. and Reps, T., “Program integration for languages with procedure calls.”, ACM Transactions on Software Engineering and Methodology 4, Vol. 1, pp.3–35, 1995.

[11] K.B.Gallagher, “Using Program Slicing in Software maintenance”, IEEE Transactions on Software Engineering, Vol. 17, No. 8, pp. 751–761, 1991.

[12] Shigeta Kuninobu, Yoshiaki Takata, Hiroyuki Seki and Katsuro Inoue”, “An Efficient Information Flow Analysis of Recursive Programs based on a Lattice Model of Security Classes”, Proceedings of Third Internatinal Conference on

Information and Communications Security (ICICS 2001), Xian, China, Lecture Notes in Computer Science 2229, pp. 292–303, 2001.

[13] 植田, 練, 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.

[14] 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.

[15] 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.

[16] 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.