平成21年度 卒業論文

論文題目

推論による情報漏えい防止のための オブジェクト関係の視覚化

神奈川大学 工学部 電子情報フロンティア学科

学籍番号　200602840

鈴木　遼

指導担当者　木下宏揚　教授

目 次

図 目 次

表 目 次

第1章

序論

現在, ネットワークの拡大などにより, 以前は紙などで管理されていたデータ群などがデジタルデータに移行されるなどして大量のデータを小さい媒体で持ち歩くことが容易になったことがあり, これによって企業の顧客データの流出といった情報漏えいによる被害が以前により増大している. こういった情報漏えいが発生する主な原因は,

- 管理していた情報データが持ちだされてしまう
- インターネット上からアクセスされて, 情報データが漏えいする
- 情報管理における内部の管理・監視体制の不備および未構築

などといった基本的には自身のセキュリティ強化を怠った結果によるものが多い.[1] また, こういった情報漏えいは主に情報そのものが丸ごと漏えいに対する対策というものは主に流出された困る情報そのもの（秘密情報）がそのまま丸ごと不正利用者に情報が流出するのを防ぐことを主として考えている. その中の一つとして CovertChannel[2][3][4] というものがある,CovertChannel は本来意図されている通信経路とは別に通信利用者にとっては想定されていない通信経路のことであり, これに対しての分析, 解析なども現在進められている. だが, それ以外にも一つ一つの情報がそれぞれ秘密情報として扱われていなくても, その情報が複数集まり, ある条件下におけるとき, 推論によって本来は得ることが出来ない秘密情報として利用者に流出してしまい情報と情報のつながりから一つの CovertChannel に近い経路が発生する可能性が出てきた. このことから, どのような情報が複数集まることによって, どういった情報が推論によって得られることができるかを考慮する必要性が生じてきた. そこで, 本

研究では情報（オブジェクト）同士の関係を考慮するためのデータ構造を提案する. また, その関係を視覚化することで, どの情報からどの情報が推論できるかを容易に把握することにより想定外の推論による情報漏えいを防ぐためのツールを実装することを目的とする.

第2章

基礎知識

2.1　アクセス行列

アクセス行列とは主体 (Subject) と客体 (Object) の関係を表した行列のことで，主体と客体の関係にはR(Read:読み込み可), W(Write:書き込み可能),RW(Read+Write:読み書き可能), φ (Phi:読み書き不可) の 4 種類の権限がある.

	S1	S2
O1	ϕ	W
O2	R	R

図 2.1　アクセス行列

2.1.1 主体（Subject）

主体とはネットワークやデータベース内で管理されている客体にアクセスする行為者であり, ユーザーに相当する.

- 名前……主体の名前
- 競合……管理しているコミュニティの情報
- 階層……コミュニティで指定されたセキュリティレベル
- 役割……客体の権限を決定する役割

2.1.2 客体（Object）

客体はネットワークやデータベース内で管理されている情報であり, ファイルに相当する.

- 名前……客体の名前
- 競合……管理している主体のコミュニティの情報
- 階層……コミュニティで指定されたセキュリティレベル
- 所有……管理している主体の情報
- プライベート……管理している主体の情報

2.1.3 コミュニティ（Community）

コミュニティとは, コミュニティの属性, コミュニティに属する主体, および, コミュニティが管理する客体とその属性の集まりから成る社会システムに相当する. コミュニティ(Community) 同士には利害関係があり, 管理している主体には組織的に階層レベルや役割を割り振られる. コミュニティにも様々な種類があるが, インターネット上のコミュニティを考えてみると主体の役割や利害関係, 或いはプライベートな情報（個人情報）が複雑に絡み合っている. 現在, 求められているのはこのように複雑に絡み合ったコミュニティにおいて実現するセキュリティモデルである. それが実現されたのが Community Based Access Control Model である.

2.2 Covert Channel

2.2.1 間接情報フロー

Covert Channel とはアクセス行列において, 本来客体（Object:データやそれを含む情報）に直接アクセスする権限（Permission:アクセス権）がない主体（Subject:利用者, ユーザー）なのにも関わらず, アクセス権限を持つ第三者の力を借りて間接的にその客体できるようになってしまう. その時に発生する客体に対しての主体へのアクセス権限が矛盾した不正な経路を Covert Channel という. またこれを間接情報フロー [5] と呼ぶ. 以下の流れが例である. 初期状態 S2 は直接 O1 の情報を読み込むことができないが以下の流れで読むことが出来てしまう.

1. S1（Subject）が O1（Object）を読み込む
2. S1 が O1 で読み込んだ情報を O2（Object）に書き込む
3. S2（Subject）が O2 を読み込む
4. 発生した Covert Channel より読めないはずの O1 の情報を S2 が読める

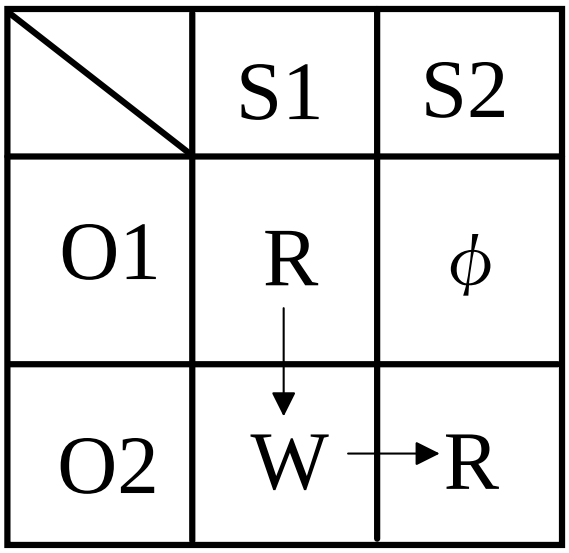

図 2.2 Covert Channel の例

このような流れで不正な情報流出が発生してしまうためアクセス制御を行う推論エンジンとしては出来る限りこれが発生するのを防ぐ検出と訂正を的確に行えるようにするのが情報フィルタに必要とされる機能である.

2.2.2 実際に発生する Covert Channel

不正な情報経路である Covert Channel を全て塞いでしまえば安全なシステムを構築することが出来るように見えるが, 単独では隠れチャンネル (Covert Channel) が存在しないようなコンピュータでもネットワークに接続されたコンピュータ群が協調することによって, 隠れチャンネルを構成できてしまう. つまり, 単独では安全なコンピュータでも, それがネットワークを構成すると安全ではなくなるような状況が簡単に存在し得るのである. このようなネットワーク構成機能の問題点が Covert Channel で利用される. 例えば以下のような例が挙げられる.

- 会社の機密データを社外へ持ち出したり, 社外の人間（社外の PC）でも見れるようにする
- mixi[7] 等の SNS の個人データが掲示板やブログ等不特定多数へ流出
- スパイウェア等, 個人 PC から情報を持ち出すためにこれを用いて通信を行い, 検知を困難とする

WWW 等, 不特定大多数が利用するネットワークでは意図しなくても Covert Channel が発生してしまう恐れがあるのでそういった情報網では比較的安易に情報漏洩が起こりうる. このように Covert Channel は今のネットワーク社会にとって情報を安易に流出させてしまう存在なのである.

2.2.3 Community Based Access Control Model

Community Based Access Control Model [2] は Covert Channel の制御を実現するためのセキュリティモデルの１つである.Access Control Agent System がこのモデルには組み込まれていて, コミュニティを用いた Covert Channel 分析が行なえる. まずユーザ数とファイル数を抑制し, 整理するために共通する属性を持った小規模なユーザの集合を Community と定義する. 各コミュニティではそれぞれ内部で Covert Channel 分析を行い,Covert Channel がおきないように制御する. 外部コミュニティと通信した場合の Covert Channel 分析は自コミュニティ, アクセス要求者, その要求について全て分析する.Covert Channel 分析は,Subject や Object の関係を以下のようなアクセス行列で表現し,Covert Channel の検出をする. 属性はアクセス制御や Covert Channel との関連性から競合, 所有, 階層, 役割, プライベートの５つを使用する. このとき, アクセス行列

から図1 のような2 × 2 行列のパターンを全て取り出し,Covert Channel 分析を行なう. このやり方により,2 × 2 行列全てのパターンだけでなくそれらを組み合わせることで3 × 3 やそれ以上の場合も Covert Channel を全て検出することが出来る. このようにこのモデルは Covert Channel を防ぐのに元々適したモデルであるので, この推論機能をベースとして改良し, 新たな処理のアクセス制御を行なう.

2.2.4 セキュリティモデル

セキュリティモデル [2] は, アクセス制御システムを構築する上で, セキュリティポリシーを具体的な論理的形式で表現したものである. そこには制御したいサービスや組織構造が反映される. 最も単純な型では,permission(read,write, ¬ read, ¬ write) であり,Subject（主体）,Object（客体）を含めた３つでアクセストリプルと呼び, それをシステムで如何扱うかによってアクセス制御が行われる. 従来のセキュリティモデルには様々な種類があり, 使用される状況に応じて使い分けたり, 複数使用する等しているが今回の実験ではアクセストリプルの構造や Covert Channel の検出・訂正を考慮したモデルとして Community Based Access Control Model を改良する. このモデルには幾つかのモデルには無い機能を持ち, 他のモデルと併用することで Covert Channel の検出機能及び今回追加する訂正機能を実行することができる.

2.2.5 情報フィルタ

情報フィルタとはCovert Channel検出時にそのCovert Channelが無くなるように特定の権限を変更することである.情報フィルタには4種類の方法があり,それぞれ一長一短がある.3種類はフロー経路の権限を禁止して遮断するのに対し,Read権限を許可する方法は情報共有の拡大の意味を持つ.以前は読めなかった客体が修正により普通に読めるようになれば,不正経路ではなくなるのでCovert Channel自体は無くすことができる.情報フィルタの具体的な処理を以下にまとめていく.図のようにCovert Channelが発生して検出された場合,以下,図2.3の(a)(b)(c)(d)のいずれかを適用すればCovert Channelが解消される.

1. (S1,O1)のREAD権限を削除
2. (S1,O2)のWRITE権限を削除
3. (S2,O1)にREAD権限を添付
4. (S2,O2)のREAD権限を削除上記のどの情報フィルタを選択するかは各コミュニティのセキュリティポリシーや主体のアクセス履歴,ユーザがどういう方針で処理するか定めるユーザーポリシーを考慮して決定するが,(a)から(d)のどの場合でもCovert Channelは訂正できる.

	S1	S2
O1	ϕ	R
O2	ϕ	RW

(a)

	S1	S2
O1	ϕ	ϕ
O2	R	RW

(b)

	S1	S2
O1	ϕ	R
O2	R	W

(c)

	S1	S2
O1	R	R
O2	R	RW

(d)

図 2.3 情報フィルタ

2.3 推論

ここでは，まず Prolog に関する基礎知識として記号論理について述べる. そして，Prolog について解説し，Community Based Access Control で用いられる推論について述べる.

2.3.1 命題論理

(propositional logic) である. 1 つの命題 (proposition) は真 (true) または偽 (false) のどちらかの値を持つ. 命題とは, 例えば「太郎は父親である」といったような事実を表したものであると考えればよい. 1 つの命題を 1 つの記号, たとえば英文字の p によって代用するものとする. 1 つまたは複数の命題を論理演算子 (logical opertor) で結合してできる式を論理式 (logical formula) という.A,B をそれぞれ論理式とすると,. 以下のような性質を表すことが出来る.

- 否定 (negation) ¬ A (not A, ∼ A)
- 論理積 (conjunction) A ∧ B (A and B)
- 論理和 (disjunction) A ∨ B (A or B)
- 含意 (implication) A ⊃ B (A → B, A ⇒ B)

これらは論理式と論理式の関係であるが，これら自体も論理式である.

2.3.2 述語論理

述語 (predicate) とは, 例えば「鳥は空を飛ぶ」のように何かの関係を表現するものである. 述語は関数のように扱うことができ，引数をとれる. 引数としてとるものを項 (term) と言う．項の個数が n だとすれば，その述語は n 項述語であると言える. 例えば ”Taro drinks water. ” という文は，

- drinks(Taro, water)

という式で表せる．この中で drinks は 2 項述語, Taro と water は項である．変数 (variable) も項である. このような式を原子論理式 (atomic formula) という．また，述語論理においては，例えば，ある特定の固体 x のもつ述語的性質 P(x) が真であるかどうかを聞くだけでなく，集合 D のすべての要素 x について P(x) が成立するかどうかを聞くことが出来る. これを，

- ∀ xP(x)

と書く. これは集合 D の ”すべての x について，その x は性質 P を持つ ”という命題である. ∀ のことを全称記号（universal quantifier）という. この論理式の真偽を決めることは，定義域 D が有限の場合には可能であるが，D が無限の場合には困難な問題となる.「すべての人間には親がある」の論理表現については以下のようになる.

- ∀ x(Man(x) → Parent(x))

となる. そして，”すべての ”という概念の対になる ”少なくとも一つは存在する ”という概念もある．これは，∃ を用いて表現され，存在記号（existential quantifier）と言う.「愛する事が出来る人がいる」というのは，

- ∃ x Love(x)

と表すことができる. ∀，∃ の両者を量記号，または限量子（quantifier）と言う. 上式はある固体 a ∈ D であって，Love(a) が真であるということであり，その場合に ∃ x Love(x) は真となる.

2.3.3 Horn 論理

Prolog については，後で述べるが，述語論理の特殊形である Horn 論理 (Horn logic) について簡単に触れる. Prolog は述語論理の特殊形である Horn 論理に基づいている.Horn 論理では, 扱う論理式をつぎのような形式のものに限定する.

- p1 ∧ p2 ∧ ⋯ ∧ pn ⊃ q

ここで p1 , p2 , ⋯ , pn , q は原子論理式または原子論理式に否定がついたものでありリテラル (literal) と呼ぶ. このような論理式をホーン節 (Horn clause) という. Prolog のプログラムはホーン節の集合からなる. 節のホーン節を Prolog では, 次のように記述する.

- q :- p1 , p2 , ⋯ , pn.

”:- ”は右辺を前提, 左辺を結論とする含意を表す 2 項演算子である. 左辺を頭部 (head) , 右辺を本体 (body) と呼ぶ. 1 つの節は必ずピリオド ”. ”で終る. 頭部あるいは本体を省略した節は特殊な意味をもつ.

- 規則 (rule) q :- p1 , p2 , ⋯ , pn.
- 事実 (fact) q.
- 目標 (goal) :- p1 , p2 , ⋯ , pn.

規則は， p1 かつ p2 かつ... かつ pn ならば q，事実は q は事実であるという事を表し，目標は p1 かつ p2 かつ... かつ pn でないかどうか調べよという事を意味している. 「事実」は頭部のみからなる節であり, このときは ”:- ”を省略する. 一般に Prolog のプログラムは, 複数の規則と複数の事実, そして 1 つの目標から構成され, 実行は, 与えられた規則と事実から目標が導けるかどうかを証明するという形で行なわれる.

2.3.4 Prolog

Prolog とは，論理学を基にして作られたプログラミング言語であり，述語表記されたものを処理させる事が出来る．Prolog は，物事の間の関係（属性）を定義していく事で，物事を抽象化し，問い合わせをする事により，結論を導きだす．この物事の関係を事実と言う．例えば「正弘は太郎の父親である」という関係があり，prolog で表すと以下のようになる.

father.pl 3

```
isFatherOf(masahiro,taro).
```

ここで，以下のような関係を定義する．

parent.pl 3

```
male(masahiro).
male(taro).
female(hanako).
isFatherOf(masahiro,taro).
isMotherOf(hanako,taro).
isParentOf( X, Y ) :- isFatherOf( X, Y ).
isParentOf( X, Y ) :- isMotherOf( X, Y ).
```

このような関係が与えられた時に,「太郎の父親は誰か」という質問をしてみる．すると以下のような結果になる．

output of father.pl

```
1 ?- isFatherOf(X,taro).
X = masahiro ;
No
```

Prolog の変数というのは，何が入っているのかは決まっていない．Prolog は質問に対して答える時に，その質問に一致する事実があるかどうかを調べる．このことをパターンマッチングといい，これにより変数の値が決まる．この場合，”isFatherOf(X,taro). ”が質問になる．この意味は，taro の父親に当てはまる X があればそれを出力するということである．つまり，Prolog はパターンマッチングする X を探す．parent.pl の ”isFatherOf(masahiro,taro). ”で「taro の父親は masahiro」であることが定義されいてるため, ”X=masahiro ; ”が結果として返る．この X を自由変数といい，この自由変数にデータが代入される．次は，二つの変数 X,Y を用いた時の例を示す．

output of father.pl

```
1 ?- isParentOf(X,Y).
X = masahiro Y = taro ;
X = hanako Y = taro ;
No
```

質問は ”isParentOf(X,Y). ”であり，親子関係がある X,Y を出力させるというものである．まず，X について当てはまるのは ”isParentOf(X, Y) :- isFatherOf(X, Y). ” より masahiro であり，X が masahiro の時にパターンマッチングするのは taro である．そして，もう一つの関係である ”isParentOf(X, Y) :- isMotherOf(- X, Y). ”より，X が hanako のときパターンマッチングするのが，taro なので，この二つが結果として返される．

2.3.5 Community Based Access Control に於ける推論

Community Based Access Control で実現される推論機能とは，情報フィルタに組み込まれる推論エンジンによって実現される．推論エンジンは，Subject 及び Object に属性を付与し，演繹推論規則を用いることでアクセスルールと照合し権限を決めるという機能を持つ．つまり，まだ権限の決められていないアクセス行列中のブランクの権限を決める事が出来る．また，異なった属性から推論された権限の矛盾や重複の訂正，すでに権限が与えられている場合の権限の重複や矛盾も検出し訂正する事が出来る．この場合の訂正は以下の 3 つ方法が考えられる．

- 競合・プライベート・階層属性を優先する
- ユーザ側に問い合わせる
- 決められたセキュリティポリシーに適した権限を矛盾した場合に自動推論

今回は，競合・プライベート・階層属性を優先にする事で実現している．これは，Covert Channel を検出を目的としたセキュリティポリシーにしているためである．これにより，Community Based Access Control で，問題にしている Covert Channel の検出も可能とする．別の章で後述するが，実際にこの機能は，Prolog によって実現した．

2.4 データ構造

与えられたデータを, 計算の高速化のために構造をもたせて記憶する手法の総称. 目的に合わせ, あるいくつかの機能を, 高速, あるいは省メモリで行えるよう設計される. 例えば, ある言葉を n 個の言葉の辞書データから検索するとき, 通常の配列では, 最悪で挿入・削除に $O(n)$, 検索に $O(\log n)$ か, 挿入と削除に $O(1)$, 検索に $O(n)$ の時間を要する. 二分探索木というデータ構造は, これらの機能を $O(\log n)$ 時間で行い, 使用メモリは $O(n)$ である.[8] 主に扱われるなデータ構造は以下の通りである.

- 連結リスト
- 配列
- スタック
- キュー
- 木構造
- ハッシュテーブル
- ルックアップテーブル
- グラフ

2.4.1 連結リスト

連結リスト（れんけつリスト, 英: Linked list）は, 最も基本的なデータ構造の一つであり, 他のデータ構造の実装に使われる. リンクリスト, リンクトリストとも表記される.

一連のノードが, 任意のデータフィールド群を持ち,1 つか 2 つの参照（リンク）により次（および前）のノードを指している. 連結リストの主な利点は, リスト上のノードを様々な順番で検索可能な点である. 連結リストは自己参照型のデータ型であり, 同じデータ型の別のノードへのリンク（またはポインタ）を含んでいる. 連結リストは場所が分かっていれば, ノードの挿入や削除を定数時間で行うことができる（場所を探すのにかかる時間はリスト上の順番の条件などにも依存するし, 後述する片方向リ

ストなのか双方向リストなのかにも依存する）. 連結リストにはいくつかの種類があり, 片方向リスト, 双方向リスト, 線形リスト, 循環リストなどがある.

連結リストは多くのプログラミング言語で実装可能である.LISP や Scheme といった言語は組み込みでこのデータ構造を持っていて, 連結リストにアクセスするための操作も組み込まれている. 手続き型やオブジェクト指向型の言語（C 言語,C++,Java）では, 連結リストを作るには mutable（更新可能）な参照を必要とする.

2.4.2 配列

配列（はいれつ,Array）は, プログラミングにおけるデータ構造の一つ. 科学技術計算分野ではベクトルという場合もある.

配列なのはデータの集合であり, 添え字でインデックスされたものを指す. 古典的なプログラミング言語では同じデータ型の集合に限定されるが, 比較的新しい言語や多くの高水準言語では異なった型も格納することができる. 通常, 変数には 1 つの値しか格納できない. しかし, ときには, ある関係を持つ複数の値を格納できる変数があると都合の良いことがある. その場合に用いられるのが配列である. 例えば,6 人の生徒の平均点を計算するプログラムを書くとする. それぞれの生徒の点数を格納する変数は, 愚直に考えれば, 次の C 言語で書かれた例のように個別に宣言することになるだろう.[6]

2.4.3 スタック

スタックは, ノード（何らかのデータを持ち, 別のノードを指し示すことができる構造）のコンテナ（データを集めて格納する抽象データ型の総称）であり,2 つの基本操作プッシュ(push) とポップ (pop) を持つ. Push は指定されたノードをスタックの先頭（トップ）に追加し, 既存のノードはその下にそのまま置いておく. Pop はスタックの現在のトップのノードを外してそれを返す. よく使われる比喩として, 食堂にあるバネが仕込まれた台に皿や盆を積み重ねておく様子がある. そのようなスタックでは利用者は一番上（トップ）の皿だけにアクセスすることができ, それ以外の皿は隠されている. 新たに皿が追加される（Push される）と, その新しい皿がスタックのトップとなり, 下にある皿を隠してしまう. 皿をスタックから取る（Pop する）と, それを使うことができ, 二番目の皿がスタックのトップとなる. 二つの重要な原則がこの比喩で示され

ている. 第一は後入れ先出し (LIFO: Last In First Out) の原則である. 第二はスタックの中身が隠されているという点である. トップの皿だけが見えているため, 三番目の皿がどういうものかを見るには一番目と二番目の皿を取り除かなければならない.[6]

2.4.4 キュー

キュー (queue), あるいは待ち行列はコンピュータの基本的なデータ構造の一つ. データを先入れ先出し (FIFO: First In First Out) のリスト構造で保持するものである. キューからデータを取り出すときには, 先に入れられたデータから順に取り出される. キューにデータを入れることをエンキュー (Enqueue) , 取り出すことをデキュー (Dequeue) という. プリンタへの出力処理や, ウィンドウシステムのメッセージハンドラ, プロセスの管理など, データを入力された順番通りに処理する必要がある処理に用いられる.

キューの変形として, 先頭と末尾の両端から入出力を行えるものを両端キュー (double-ended queue, deque) という.

キューとは逆にLIFO（後入れ先出し）のリスト構造を持つデータバッファをスタックと呼ぶ.[6]

2.4.5 木構造

木構造は, ノード（節点, 頂点）とノード間を結ぶエッジ（枝, 辺）あるいはリンクで表される. ノードには何らかのデータ（値, 条件, 別のデータ構造, 別の木構造）が付属している. 木構造内の各ノードは, 0 個以上の子ノード（child nodes）を持ち, 子ノードは木構造内では下方に存在する（木構造の成長方向は下とするのが一般的である）. 子ノードを持つノードは, 子ノードから見れば親ノード（parentnode）である. 同じ親を持つノード同士を兄弟という. ノードは高々1 つの親ノードを持つ. 最底辺の子ノードから, あるノードまでのエッジ数を, そのノードの「高さ」という. （後述する）根ノードの「高さ」は, 木構造の「高さ」である. 逆に根ノードから最底辺に向かってのエッジ数を「深さ」という.

木構造の頂点にあるノードを根ノード（root node）と呼ぶ. 頂点にあるため, 根ノードは親ノードを持たない. 木構造に対する各種操作は, 一般に根ノードを出発点とす

る（アルゴリズムによっては, 葉ノードから開始して根ノードに到達して完了するものもある）. 他のノードにはエッジあるいはリンクを辿ることで必ず到達できる. 形式的定義では, そのような（根から特定ノードまでの）経路は常に一意である. 図で示す場合, 根ノードが一番上に描かれるのが普通である. ヒープなどの木構造では, 根ノードは特別な属性を持つ. 木構造内の全てのノードは, そのノードを頂点とする部分木の根ノードと見なすことができる.

木構造の最底辺にあるノードを葉ノード（leaf node）と呼ぶ. 最底辺にあるため, 葉ノードは子ノードを持たない.

内部ノード（internal node,inner node）は, 子ノードを持つノード, すなわち葉ノード以外のノードを意味する.[6]

2.4.6 ハッシュテーブル

ハッシュテーブルはキーをもとに生成されたハッシュ値を添え字とした配列である. 通常, 配列の添え字には非負整数しか扱えない. そこで, キーを要約する値であるハッシュ値を添え字として値を管理することで, 検索や追加を要素数によらず定数時間 O(1) で実現する. しかしハッシュ関数の選び方（例えば, 異なるキーから頻繁に同じハッシュ値が生成される場合）によっては, 性能が劣化して最悪の場合 O(n) となってしまう.[6]

2.4.7 ルックアップテーブル

ルックアップテーブル（Lookup table）とはコンピュータにおいて, 効率よく参照や変換をする目的でつくられた配列や連想配列などのデータ構造のことをいう. 例えば大きな負担がかかる処理をコンピュータに行わせる場合, あらかじめ先に計算できるデータは計算しておき, その値を配列（ルックアップテーブル）に保存しておく. コンピュータは配列から目的のデータを取り出すことによって, 計算の負担を軽減し効率よく処理を行うことができる. またあるキーワードを基にあるデータを取り出すとき, その対応を表としてまとめたものもルックアップテーブルといえる.[6]

2.4.8 グラフ

グラフ理論とは数学の 1 分野で節点（Node）の集合と辺（Edge）の集合で構成されるグラフ性質について学問である.
グラフとは例えば電車の乗換案内図を考える際には, 駅（ノード）がどのように路線（エッジ）で結ばれているかが問題であって, 線路が具体的にどのような曲線を描いているかは本質的な問題でないことが多い. 事実, 乗換案内図を書く場合には, 駅間の距離や微妙な配置, 路線の形状といったものは, 地理的な実際のそれとは異なって描かれることが多い.（例えば, 実際には外回りが左にカーブしている場所のある山手線が楕円形を描いているなど）電車で移動する人を対象とした乗換案内においては, 駅と駅の「つながり方」が主に重要なのである. このように,「つながり方」に着目して抽象化された「点とそれをむすぶ線」の概念がグラフであり, グラフが持つ様々な性質を探求するのがグラフ理論である. つながり方だけではなく「どちらからどちらにつながっているか」をも問題にする場合, エッジに矢印をつける. このようなグラフを有向グラフという. 矢印のないグラフは, 無向グラフという.[6]

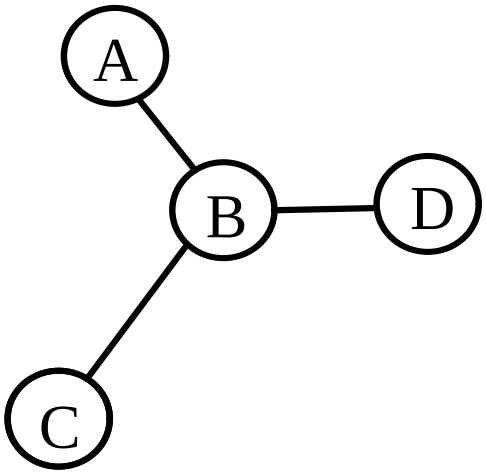

図 2.4 頂点が 4, 辺が 3 からなるグラフ

2.5 グラフの定義

有向グラフ

集合 V,E と E の元二つの V を元の対で対応させる写像.

f:E → V × V
の 3 つ組
G:=(f,V,E)
を有向という.V の元を G の節点 (Node) またはノード,E の元を G の辺 (Edge) またはエッジと呼ぶ.

無向グラフ

P(V) を V の冪集合とする.E の元に V の 部分集合を対応させる写像
g:E → P(V) があって,E の任意の元 e の像が g(e) = v1, v2 のようにちょうど二つの元の集合になっているとする. このとき, 三つ組 G:=(g,V,E)
を無向グラフという.V の元を G の頂点,E の元を G の辺と呼ぶ.g の値が常に k ¿ 2 個の元の集合となっているとき,k-ハイパーグラフという. E を最初からある集合の部分集合と考えれば, 写像を用いずにグラフを定義することもできる. 有向グラフでは,E を V × V の部分集合, 無向グラフでは,E を P(V) の部分集合で, 二つの元の集合のみからなるものとすればよい.

2.5.1 表現の選択

グラフを実際に表現するための主なデータ構造として,2 種類のデータ構造がある. 第一は隣接リストと呼ばれるもので, 各ノード毎に隣接するノードのリストを保持するデータ構造である. 第二は隣接行列と呼ばれるもので, 行と列にエッジの始点と終点となるノードが並んだ 2 次元の配列で表され, 配列の各要素は 2 つのノード間にエッジがあるかどうかを示す値が格納される. 隣接リストはまばらなグラフに適しており, そうでない場合は隣接行列の方が望ましい. なお, 非常に大きなグラフでエッジに何らかの規則性がある場合, シンボリックグラフという表現も選択肢としてありうる.

2.5.2 他データ構造との比較

グラフデータ構造は階層的ではないため, 個々の要素が複雑に絡み合っているようなデータを表現するのに適している. 例えば, コンピュータネットワークはグラフとして表現するのに適している. 階層的なデータは二分木や二分木以外の木構造で表される. ただし, 木構造はグラフ構造の特殊ケースと見ることもできる.

第3章

推論による情報漏えい

3.1　推論攻撃

今回は推論による情報漏えいの例として推論攻撃というもの説明がある. 推論攻撃 [9] とは, 利用者が現時点で利用を許可している問い合わせとその実行結果から, 本来は許可されていない問い合わせの実行結果（秘密情報）を推論により入手しようとすることであり, これは主にデータベースセキュリティの管理などで取り入れられている. 今回では秘密情報として扱われているオブジェクトに対して, 本来そのオブジェクトの情報を得られない利用者が現時点で得られるオブジェクの情報から秘密情報を推論により把握してしまうことを考える. また, こういった推論攻撃は推論攻撃によりどのような情報が得られるかが一般に明らかにはできないので, 情報の管理者などは推論攻撃が成功する可能性をあらかじめ考慮し, その推論を制御することが重要である. ネットワーク上においてオブジェクトとオブジェクト同士に直接の繋がりがなくても何らかの推論規則を持つことで, 秘密情報を推論できてしまう可能性がある. 利用者が推論によって新たな情報得るために必要な情報は, 例えば, 図 3.1 に示してある構図だと名前, レポート, 評価方法の三つの情報を持っていてそれに対応する推論規則を持っているとき一人一人のレポートの成績（秘密情報）が推論により分かるというものであるが, このとき名前, レポート, 評価方法のオブジェクトから成績というオブジェクトの情報が推論されるということになり, それをグラフ構造にして表してある.

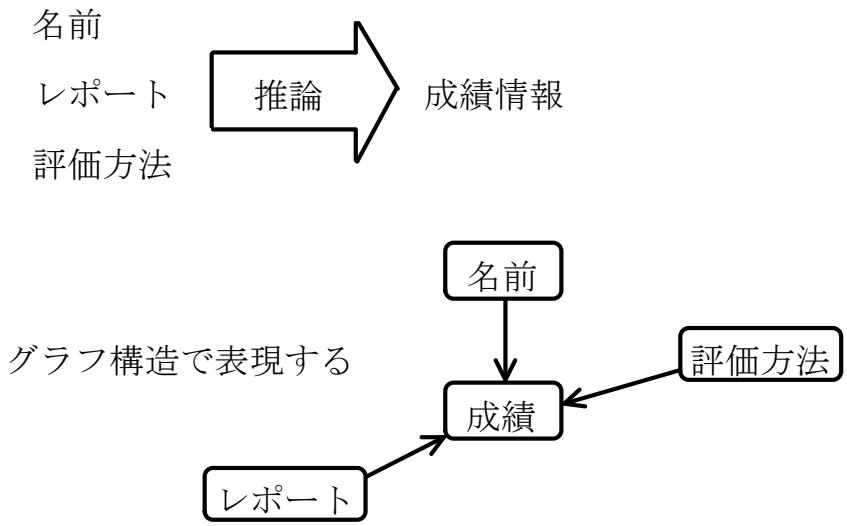

図 3.1 グラフ構造の簡単な例

第4章

提案ネットワーク

4.1　オブジェクトの視覚化

4.1.1　視覚化の形式

安全なオブジェクトの関係から意図しない経路が発生するには以下の事柄が必要である

- 利用者がオブジェクトを入手する
- 推論するための条件（ルール）を利用者が持っている

このことからオブジェクト一つ一つを Node, オブジェクト同士に存在する関係を（Edge）として考えたグラフ構造を用いた視覚化を提案する. グラフ構造を提案する主な理由はグラフ構造は他データ構造と違い階層的ではないため, ネットワーク上にあるオブジェクト同士の要素が複雑に絡み合うことになるので階層的なデータ構造よりグラフ構造のほうが適している. また図 4,1 に示しているが, グラフ構造の図に方向性を持たせることによって「一つのオブジェクトの情報を得るためには他のどのオブジェクトから情報を得ると推論がされる」という関係を, 図とは別に表として表すことが可能となりオブジェクトを纏めやすくなるからである.

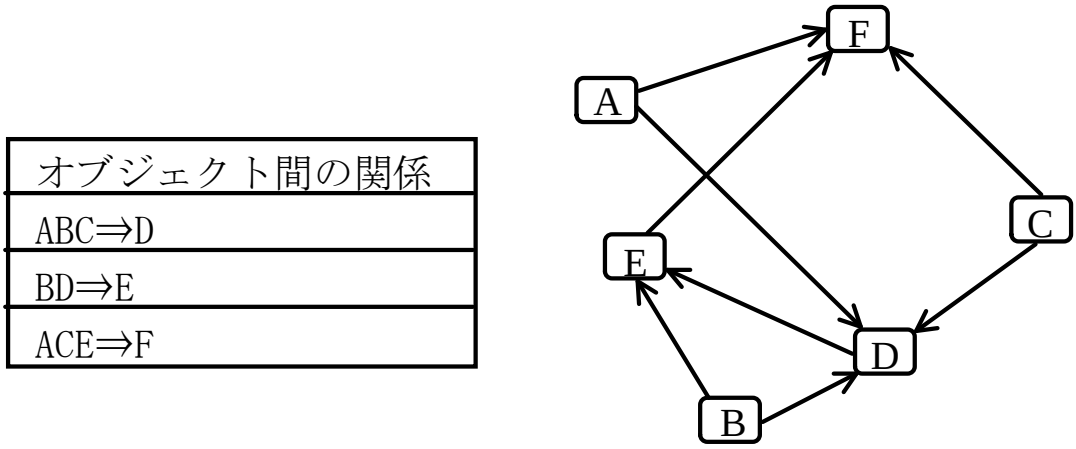

図 4.1 グラフ構造とオブジェクト関係の表

また視覚化の実装に際して以下のことが予想される.

1. オブジェクトの関係をまず視覚化
2. その中から秘密としておきたいオブジェクトをクリックなどで設定
3. その後推論などにより情報漏えいが発生したときに自動検出

それの例としたものを図 4,2 に示してある. また, それ以外にも 4,3 に示してあるよう

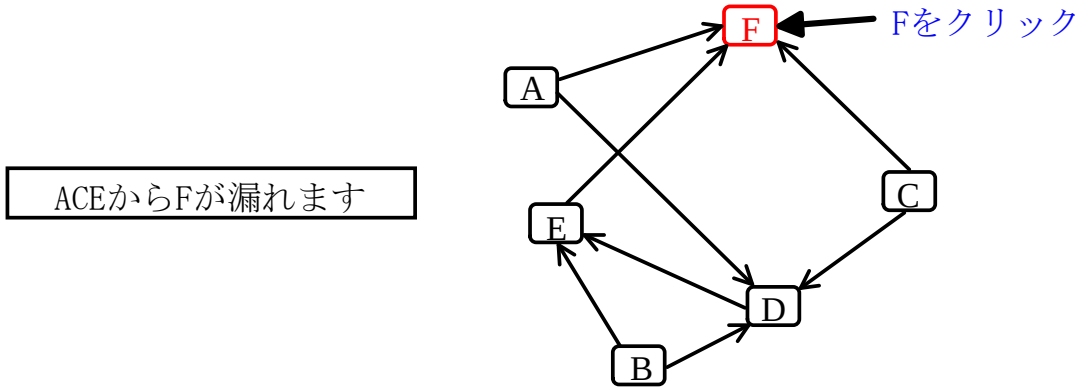

図 4.2 視覚化の具体例 1

に, 例えば成績というオブジェクトを秘密情報としたときに, 名前, 評価方法, この二つのオブジェクトの情報を得た時に残りのレポートというオブジェクトの情報をこのまま本来成績のオブジェクトを見る権限がない利用者が入手してしまうと, 成績のオ

ブジェクトが推論によって見つかってしまうため, 意図的にレポートというオブジェクトを見れなくし, さらにそのことを事前に利用者に教えてくれるといった流れ.

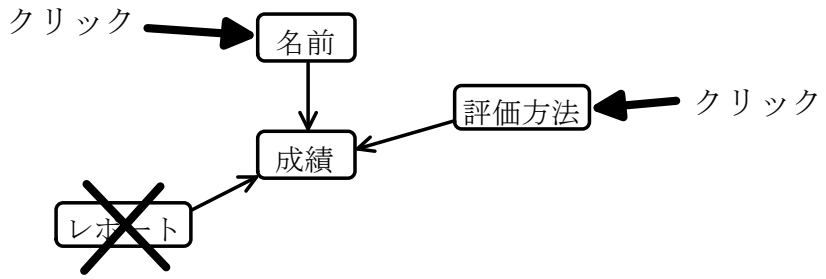

図 4.3 視覚化の具体例 2

またこの関係図を述語論理で表すことができる, 上記にある「名前, レポート, 評価があれば成績が判る」のとき, 名前, レポート, 評価, 成績の四つのオブジェクトをそれぞれ nam,rep,est,res, として表した時下の式となる

- res(nam,rep,est)

この場合,res が三項述語,nam,rep,est の三つが項となる.

4.1.2 視覚化に関して

今回のシステムは主にネットワークの利用者を対象として考える. また, ネットワーク上での利用を考慮して, オブジェクト間の情報（Node の全体図）, またそのオブジェクト間の繋がりを示す情報（Edge の情報）やオブジェクトの推論に必要なルール（推論規則）などのデータ群は外部から入力することができるようにする. 本研究では Node をクリックしたときに Node に変化をつけられるようにすることを目標とする.

4.2 提案システムの動作

今回, システム作成に使用したソフトは AdobeFlex3(ActionScript3.0) を使用. また先で述べたオブジェクト間の情報を XML 形式で記述する. 本研究では, オブジェクト関係の視覚化を重視しているため, オブジェクト間の情報, オブジェクトの推論に必要な推論規則などの外部ファイルは予め与えられているものとして, システムを提案する. 下にプログラムの一部と実行結果の図を下記に記述する.

```
import Graph;
import Node;
import Edge;
import mx.core.UIComponent;
import flash.display.Sprite;
import flash.events.Event;
import flash.events.*;
import flash.net.URLLoader;
import flash.net.URLRequest;
import flash.text.*;

public var graph:Graph = new Graph();
public var abc:UIComponent = new UIComponent();
public function sample(event:Event):void{
addChild(abc);
abc.addChild(graph);
var a1:Node = graph.createNode(250,250,{text:member.main});
for (var i:Number=0;i<member.sub.length;i++){
var   n:Node=a1.add(100+Math.floor(Math.random()*i*100),
100+Math.floor(Math.random()*i*100),
{color : Math.floor(Math.random() * 0xffffff),text:member.sub[i]});
}
```

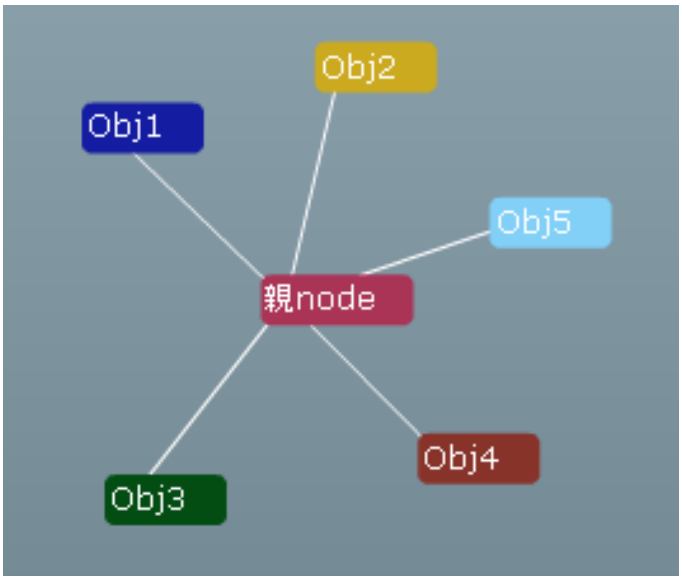

図 4.4 実行結果

第5章

結論

今回は利用者が所得しているオブジェクトとそれに関係しているなんらかのオブジェクトを所得して,尚且つ利用者がそれを結び付けることが可能な推論規則を所得しているときに,秘密情報が流出してしまうことを考慮した上でオブジェクト同士の関係を視覚化するシステムをオブジェクトを Node、オブジェクトに存在する関係を Edge を用いたグラフ構造で提案した.

あらかじめ推論規則など与えられていることを前提でシステム提案をしているがオブジェクトとそれを結ぶ関係の視覚化には成功した. 今後の課題としては推論規則自体も一つのオブジェクトとして見なすことも可能なので,もし推論規則をオブジェクトとしてみなした場合にどのように処理をすべきかの検討. そして外部ファイルとしてオブジェクト関係を入力できるかの検討などがある.

謝辞

本研究を行なうにあたり，終始熱心に御指導していただいた木下宏揚教授と鈴木一弘助手に心から感謝致します．また，公私にわたり良き研究生活を送らせていただいた木下研究室の方々に感謝致します．

2010年2月
鈴木　遼

参考文献

[1] ”情報漏洩について考えるサイト”
http://jouhourouei.j-nic.co.jp/index.html

[2] ”酒井剛典, 森住哲也, 畔上昭司, 小松充史, 稲積泰宏, 木下宏揚：“ Covert Channel 分析メカニズムと EJB による情報フィルタの構築 ”,2006 年暗号と情報セキュリティシンポジウム, (2006).”

[3] 森住哲也, 木下宏揚：“ 社会システムの中の Covert Channel について ”, 技術と社会・倫理研究会，(2005).

[4] ストレージネットワーク用語集 2008
http://www.snia-j.org/dictionary/

[5] 内野雄策:”SNS における情報漏洩を防止するための情報フィルタの適用 (2009)”

[6] ”ウィキペディア（Wikipedia)”
http://ja.wikipedia.org/wiki/

[7] ”株式会社ミクシィ”
http://mixi.jp/

[8] ”OR 事典” ”http://www.weblio.jp/cat/academic/orjtn”

[9] 高須忠和, 橋本健二, 石原靖哲, 藤原融: ”XML データベースへの型推論を用いた攻撃に対する安全性検証 (XML),(2006)”

質疑応答

Q1:発表に使われていた動画のノードが最初被っていたのは特に意味があるのか.また、動かせることのメリットは何か.(松澤教授)

A1:ノードが被っていたのはノードの配置に固定座標が決められているとは限らないので乱数を用いているため, また動かせることによるメリットは, 視覚化における見やすさの向上のためであるが, 今後は不必要になる可能性もあります.

Q2:実行図のプログラムにあるオブジェクトの中身について.(松澤教授)

A2:今のプログラムにあるオブジェクトについては純粋にオブジェクトの繋がりしか読み込んでおらず, 推論するために必要な要素を取り込んでいません.