情報処理学会論文誌　Vol. 51　No. 9　1682–1691 (Sep. 2010)

論理プログラミングを基礎とした認可ポリシ記述言語

橋本正樹[†1]　金美羅[†1]
辻秀典[†1,†2]　田中英彦[†1]

近年の情報システムでは脆弱性を完全に排除するのが難しいため，多層防御によってセキュリティ・インシデントの発生に備える必要がある．多層防御を効果的に実現するためには，細粒度の強制アクセス制御を行うための膨大なアクセス制御規則をポリシとして記述する必要があるが，既存の記述方式は可読性や保守性に問題がある．本研究では，論理プログラムとしてアクセス制御規則を記述することで，属性の継承や頻出する認可手順のサブルーチン化をサポートするポリシ記述言語を提案し，この問題を解決する．本稿では，認可判定の妥当性と表現力を評価して，本言語の有用性を実証し，期待される効果を考察する．

Policy Description Language for Authorization Using Logic-based Programming

Masaki Hashimoto,[†1] Mira Kim,[†1] Hidenori Tsuji[†1,†2] and Hidehiko Tanaka[†1]

Recently, with the impossibility of eradicating the vulnerabilities of information systems, we must prepare for the occurrence of the security incident by the multi-layer defense called Defense-in-Depth strategy. In the multi-layer defense, it is important to authorize accesses in fine-grained granularity to compose each layer effectively and many access control models are proposed to follow them. However, policy description languages proposed so far cannot express the models appropriately in proper granularity. In this paper, we propose a policy description language which can designate many kinds of conditions for access control like dynamic status of application process as an element of decision data, and implement it in Datalog. Using the proposed language, we compose the policy of SELinux which is a major implementation achieving the multi-layer defense, and we confirm the advantages of the proposed language by evaluating the validity and the expressiveness.

1. はじめに

近年の情報システムは大規模・複雑化しているため，脆弱性のないシステムを構成するのが困難である．そのため，現実的なセキュリティ対策としては，脆弱性の存在を前提としたシステム設計を行い，多層防御戦略[1)]をとることが有効である．多層防御は，情報システムを厳密に区画化しておくことで，仮にマルウェアやウィルスなどによるゼロデイ攻撃が実行され，システムへの侵入を許した場合にも，重要な情報資産への攻撃を遅延したり，被害範囲を局所化したりすることができる．

多層防御は，細粒度の強制アクセス制御によってシステム内を細かく区画化するほど有効性が高まるが，そのためには膨大なアクセス制御規則をポリシとして記述する必要がある．一方で，区画の粒度を粗くしてアクセス制御規則の記述量を低減させた場合は，セキュリティインシデント発生時の直接的な被害範囲が拡大するうえ，被害伝播の遅延効果も薄くなるため，多層防御の有効性を相殺してしまう．そのため，多層防御を効果的に適用するためには，膨大なポリシを簡潔に記述する工夫が必要となるが，SELinux[2)]をはじめとした多層防御の既存実装例では，アクセス制御規則を個々に直接記述する手法が主であるため，その粒度を細かくするほどに記述量が膨大になる傾向があり，ポリシ全体としての動作は，専用の検証用ツールを用いないことには把握が困難である．

本研究は，論理プログラムとしてアクセス制御規則を表現することで，属性の継承やサブルーチンとして構造化された記述が可能なポリシ記述言語を提案し，この問題の解決を図るものである．本言語は，個々のアクセス制御規則を記述するための宣言文と，宣言文の集合として構成されたポリシに問合せを行うための認可判定文を，構文規則とその形式的な意味，推論規則を定めることで基本設計を行い，これを Prolog[3)] のサブセットである Datalog[4)] を用いて実装している．また，本稿では，本言語を用いて具体的なアクセス制御モデルを構造的に記述する手法を示したうえで，SELinux のポリシを実際に記述した実験システムを構成し，認可判定の妥当性と表現力の評価を行った．

以下に，本稿の構成を示す．はじめに，2 章では，脆弱性の存在を前提とした戦略としての多層防御について説明し，関連研究として多層防御の実装例である SELinux とポリシ記述

†1 情報セキュリティ大学院大学
Institute of Information Security
†2 株式会社情報技研
Advanced Institute of Information Technology

に関する近年の研究動向を説明した後に，本研究の目的を説明する．次に，3 章では，本言語の基本設計として，文法と意味，推論規則を示し，Datalog を用いた実装方法を説明する．続いて，4 章では，本言語を用いてアクセス制御モデルを記述する具体的な手法を説明し，5 章では，本言語の評価実験とその考察について述べる．最後に，6 章で，本稿をまとめる．

2. 本研究の背景と目的

本章では，はじめに，研究の背景として，情報システムの保護手法である多層防御について，脆弱性との関連性に着目して説明する．また，その実装例として SELinux を取り上げ，細粒度の強制アクセス制御を用いて多層防御を実現する方式と，その際に課題となるポリシ記述について説明する．その後，ポリシ記述に関する近年の研究を示し，最後に本研究の目的を説明する．

2.1 脆弱性と多層防御

近年の情報システムは大規模・複雑化しているため，脆弱性のないシステムを構成するのが困難である．ソフトウェアの解析技術や開発手法の発展などによって，その混入を防ぐ仕組みが洗練されてきているが，情報システムへの攻撃では，最も弱い部分を能動的に探し出し，そのパスを意図的にたどって実行することが多いため，通常の運用では無視できるほどの微細な脆弱性であっても，システムを致命的な状態に陥らせる可能性がある．そのため，セキュリティの観点からすると，脆弱性を根絶できない限りは，たとえ可能性が低くても，攻撃が成功する事態を考慮する必要がある．したがって，これを考慮した現実的なセキュリティ対策としては，脆弱性の存在を前提としたシステム設計を行い，多層防御戦略をとることが有効である．多層防御戦略は，情報システムを厳密に区画化しておくことで，仮にマルウェアやウィルスなどによるゼロデイ攻撃が実行され，システムへの侵入を許した場合にも，重要な情報資産への攻撃を遅延したり，被害範囲を局所化したりするものである．

今後は，クラウドコンピューティングや Web サービス，Grid コンピューティングによって，情報システムが従来以上に大規模・複雑化していくことが予想されるため，結果として，システムから脆弱性を根絶するのはさらに困難となっていく．したがって，真に信頼できるコンピューティング環境を構築するためには，脆弱性の存在を前提としたシステムを構築することが現実的で，そのために，多層防御戦略の実現が望まれる．

多層防御を実現するためには，はじめに，層の構成方針を定めるアクセス制御モデルを決定する．アクセス制御モデルには，たとえば，RBAC[5] や DTE[6]，HBAC[7] など様々なモデルが提案されており，それぞれに利害得失があるため，実現を目指すシステムの特性に応じてアクセス制御モデルを決定する必要がある．アクセス制御モデルが決まると，次に必要になるのは，その方針に従ってアクセス制御規則を具体的に記述することで，これはポリシ記述言語により実現する．ポリシ記述言語は，アクセス制御モデルを過不足なく表現できることが必要で，記述する人間にとって直感的に理解できるものであることが望ましい．最後に必要となるのは，強制アクセス制御機構で，これは参照モニタとも呼ばれる．強制アクセス制御機構は，ポリシ記述言語によって記述されたアクセス制御規則を漏れなく確実に執行するために，全アクセスを補足して仲介する．近年では，プロセスがリソースアクセスをする場合に迂回不可能なパスとして OS を利用し，これを実現するセキュア OS が実装されている．

2.2 関 連 研 究

本節では，はじめに，多層防御を実現した実装として SELinux について説明し，これを例にポリシ記述に関する課題を示す．次に，ポリシ記述に関する近年の研究動向を説明し，本研究との関連を述べる．

2.2.1 SELinux における多層防御の実現方式とその課題

SELinux は米国家安全保障局を中心としたオープンソースコミュニティによって活発に開発が進められているセキュア OS の代表格である．SELinux は，Flask セキュリティアーキテクチャ[8] の Linux に対する実装で，セキュリティサーバとオブジェクトマネージャ，アクセスベクタキャッシュから構成される．Linux における実装では，システムコールをフックすることでアクセス制御に関連するカーネル内制御を奪い，独自のポリシによるアクセス制御を SELinux が強制する．

SELinux は，原理的にすべてのシステムコールに対してアクセス制御の判定を行えるため，RBAC モデルや DTE モデルをシステムコールレベルの粒度で実現することが可能である．しかし一方で，細粒度のアクセス制御を行うためには当然その制御規則となるポリシについても細粒度に記述する必要があるが，これは人間にとって非常に困難な仕事である．

たとえば，SELinux では以下のようなアクセス制御規則を並べてポリシを構成する．

allow *acct_t unconfined_t*:*fd use*;
allow *acct_t init_t*:*process sigchld*;
allow *acct_t init_t*:*process signull*;
allow *acct_t rpm_t*:*fd use*;

この例では，各行が 1 つのアクセス制御規則を表現しており，たとえば 1 行目では，最初の項目 *allow* によって，当該行が許可される規則であることを示す．2 つ目の項目は，この

規則を適用するサブジェクトの情報で，3 つ目以降はオブジェクトの情報と操作内容を示す．

SELinux では，上記文法によるアクセス制御規則が，デフォルト・インストールされる個人ユーザ用の簡易版ポリシでも 40,000 行近く羅列されている．そのため，たとえ 1 行 1 行の各アクセス制御規則を理解できたとしても，その総体として構成されるポリシで実際に何が起こるのかを判断するのは難しい．さらに近年では，SELinux による強制アクセス制御規則の適用範囲を OS 階層の資源から，アプリケーション層の要素や他システムの要素にまで拡大する実装が提案されているため[9),10)]，適切なポリシ記述がますます困難になることが推測できる．

2.2.2　ポリシ記述言語に関わる近年の研究状況

SDSI[11)] と SPKI[12)] は，アクセス制御を行う際に証明書の交換を行うことでシステム間の信頼性を評価するためのフレームワークである．SDSI は “speaks-for” という述語を利用して論理の構築を行い，SPKI では宣言内で述語をタグとしてエンコードすることができる．SDSI と SPKI 双方ともに，ポリシを英文に類似した構文規則によって表現できるという特徴を持つが，SDSI は “speaks-for” のみを述語として利用するため，応用に応じた述語を定義できないという制限がある．また，SPKI は複数の述語を利用可能であるが，あらかじめ定義されたタグに制約されるため，変数を扱うことができないので，こちらもやはり適切な述語を定義するのが難しいという課題がある．同様に，PolicyMaker[13)] と KeyNote[14)] においても，認可判定に必要となる様々な状態を定義することができるという特徴を持つフレームワークであるが，述語の定義に厳しい制限があるため，応用に応じた述語を利用することはできない．これらの言語は領域間の信頼性管理に主眼をおいているため，述語をある程度絞り込んでいても有効であるが，これを認可判定言語として用いた場合，現実に利用されている様々なアクセス制御モデルに対応するのが難しい．そのため，本研究では，ユーザ定義の操作内容を述部の一部として指定する構文規則を定めることで，任意の意味を持つ述部を記述可能な方式を検討した．

SecPAL[15)] は，制約論理言語によるポリシ記述言語で，認可要求は節の集合に対するクエリが成功したときに承認される．SecPAL の文法は自然言語に近く，意味付けは 3 つの推論ルールから構成されており，否定的なクエリや再帰的な述語，回数を指定可能な権限委譲やその他様々な制限をサポートすることで，多くのアクセス制御モデルを汎用的に表現可能である．また，Lithium[16)] も SecPAL 同様に，一階述語論理を基礎に推論を用いてポリシを記述する言語で，再帰的な表現を制限することにより，否定を含んだ推論を実現している．これらの言語は，論理型言語と推論によってポリシを表現する点が本研究と共通であるが，前者は特に，比較的大規模なシステムにおいて，ポリシを管理領域ごとにモジュール化して構成することを前提とし，分散管理された領域間で権限の委譲を柔軟に行うための記述方法に主眼をあてた研究で，後者は論理的な否定を形式的に正しく推論することに注力している．本研究は，認可判定規則に対する記述の抽象度を高めることで記述方法そのものの効率化を目指した点がこれらの研究と異なる．

2.3　本研究の目的

本研究の目的は，論理プログラムとしてアクセス制御規則を記述することで，属性の継承や認可手順のサブルーチン化をサポートするポリシ記述言語を提案し．これによって，ポリシ記述の複雑性の問題を，柔軟な構造化記述手法によって解決することである．

従来のポリシ記述手法は，2.2 節で例にあげた SELinux や XACML[17)] のように，個々のアクセス制御規則をそれぞれ独立に記述し，また，その各要素を直接指定するため，可読性や保守性に問題があった．この問題は，多層防御の有効性を向上させようとシステム内を細かく区画化するほどに，必要となるアクセス制御規則の粒度も細かくなるため，より大きな問題となる．その一方で，アクセス制御規則の粒度を粗くすると記述量を低減させることができるが，この場合はセキュリティインシデント発生時の直接的な被害範囲が拡大するうえ，被害伝播の遅延効果も薄くなるため，多層防御の有効性を相殺してしまう．また，今後クラウドや Grid の普及により，複数システムの連携処理が一般化し，個々のシステムで任意にアクセス制御規則を定めるのではなく，ポリシによって全体を制御するような場合には，記述量が爆発することが容易に想像できる．

そこで，これらの問題に対抗するために，近年では 2.2.2 項で示したようなポリシ記述に関する様々な研究が行われているものの，既存研究ではポリシの記述量そのものを低減させるような記述手法について検討が不十分であるため，本研究では，特にこの点に着目し，ポリシ記述の抽象度を高めて効率的に記述する手法を検討する．また同時に，プログラムとして記述したポリシが論理的に正しく動作して認可判定を行えることと，記述量を低減させることができていることを，実際に利用されているポリシを提案手法で記述する実験によって定量的に評価し，確認する．

ポリシを論理プログラムとして構造的に記述することで，数理論理学的な裏付けを自然にポリシ記述に適用できることを期待でき，これにより，たとえば，通常は別の独立した応用プログラムとして実現されているポリシの検証機能を記述方式そのものに組み込み，検証結果をアクセス制御の条件とすることでアクセス権限の競合をあらかじめ排除したり，他システムから依頼された仕事を処理する代理プロセスに対して，仕事の内容に応じた最小限の権

限セットを付与したりすることなどを期待できる．また，補助的な規則によって新規則の追加方法や既存規則の変更を制御する機能や，認可判定に利用する引数を認可判定機構が状況に応じて変更する機能なども考えられる．さらには，アクセス制御対象が特定の属性を保持しているかどうかを検証することで，たとえば「停止中である」や「汚染されている」といった動的に遷移しうる情報に基づいたアクセス制御を行い，意図しない情報流を未然に防ぐことなどが期待される．

3. ポリシ記述言語の設計と実装

本言語は，アクセス制御規則を設定するためのポリシ宣言文と，ポリシに対してアクセス可否の問合せを行うための認可判定文から構成される．本章では，はじめに，それぞれの構文規則とその意味を形式的に定義し，これに適用する推論規則を述べることで本言語の設計を説明する．その後，Prolog のサブセットである Datalog を用いた本言語の実装について説明する．

3.1 ポリシ記述言語の設計

本節では，ポリシ記述言語の設計として，はじめに，ポリシ宣言文と認可判定文の構文規則を定義し，次に，その形式的な意味と推論規則を説明する．本言語は，英語に類似した構文規則と意味，単純な推論規則によって，理解しやすいものとなるように工夫した．

3.1.1 ポリシ宣言文と認可判定文の構文規則

本言語は，ポリシを以下の構文規則で記述された宣言文の集合（DS: *Declarative Statements*）として構成し，アクセス制御の際にはポリシに対して，認可判定文（Q: *Authorization Query*）によってアクセス可否を問い合わせる．

DS ::= **Policy specifies** $fact$ **if** $fact_1$,..., $fact_n$ $(n \geq 0)$
Q ::= **Policy specifies** $fact$ | $\exists x$(Q)
S ::= E **tagged** A
V ::= **is permitted to** O S | **is forbidden to** O S | **moves to** S **with** O
I ::= **inherits** A
$fact$ ::= S V | A I | S

本言語は述語論理を基礎とし，上記のうち，ボールド書式で示されたものは構文規則の基礎的な要素である述語を表す．E，O，A は，それぞれアクセス主体とその対象，操作内容，属性を示す項である．すべての項にはそれぞれその項が属する 1 つの領域が関連づけられているとする．イタリック書式の小文字で示されたものは変数項を示し，変数項への項の代入は，変数項とその項の領域が一致したときのみ許される．また，$fact_i$ は $fact$ と同種のものであるが，それぞれを区別するために接尾辞を付したものである．S はアクセス主体や対象に属性を付したもので，本言語では S を主部としたものに述部となる V を与えることでアクセス制御規則を表現する．I はもう 1 つの述部となりうる要素で，この場合，主部は A のみとなる．各述部の直感的な意味は，“**is permitted to**” と “**is forbidden to**” によって，主部 S_1 がある対象 S_2 に対して O を許可/拒否することを示し，また，“**moves to**” によって，主部 S_1 が O を契機に S_2 に遷移することを示す．

3.1.2 構文の形式的な意味と推論規則

本言語では，宣言文の集合であるポリシ DS に対して認可判定文 Q が与えられたときに，以下に示す DS, $\theta \vdash$ Q の関係によって，認可判定文の解釈を決定する．Q の真偽はそれぞれ認可判定時の許可と拒否として解釈されるが，Q が変数項を含む場合は Q を真とするようなすべての応答が返される．θ は，変数項を別の変数項や定数項に対応付けする写像で，$fact\theta$ は，θ を $fact$ に適用したものである．また，θ_x は，$\mathrm{dom}(\theta) - \{x\}$ を領域とする写像を示し，vars(X) は，節 X に存在する変数項を示す．

DS, $\theta \vdash$ **Policy specifies** $fact$ iff DS $\models$ **Policy specifies** $fact\theta$,
and $\mathrm{dom}(\theta) \supseteq$ vars(**Policy specifies** $fact$)
DS, $\theta\_x \vdash \exists x$(Q) iff DS, $\theta \vdash$ Q

次に，宣言文と認可判定文に適用する推論規則を以下のように定義する．推論規則は，特定の構文構成に対して自動的に適用することで別の新しい構文を導くもので，宣言文の記述量を軽減することができる．推論規則 (a) は，条件節のある宣言文と，その条件節の各 $fact$ を $\mathrm{vars}(fact\theta) = 0$ とするような θ があった場合に，自動的に変数項のない $fact\theta$ を導く．なお，$\mathrm{vars}(fact\theta) = 0$ は，$fact\theta$ に変数項が存在しないことを示すものである．また，推論規則 (b) は，属性 A_2 が属性 A_1 を継承するような構文と，属性 A_1 を付された変数 x に対して V_1 を宣言する構文があった場合に，自動的に属性 A_2 を付された変数 x に対しても V_1 を宣言する構文を導く．

推論規則 (a)

$$\frac{\begin{array}{c}(\textbf{Policy specifies } fact \textbf{ if } fact_1, \ldots, fact_k) \in \mathrm{DS} \\ \mathrm{DS} \models \textbf{Policy specifies } fact_i\theta \text{ for all } i \in \{1..k\} \\ \mathrm{vars}(fact\theta)=0\end{array}}{\mathrm{DS} \models \textbf{Policy specifies } fact\theta}$$

推論規則 (b)

$$\frac{\text{DS} \models \textbf{Policy specifies } A_2 \textbf{ inherits } A_1 \quad \text{DS} \models \textbf{Policy specifies } x \textbf{ tagged } A_1\ \text{V}_1}{\text{DS} \models \textbf{Policy specifies } x \textbf{ tagged } A_2\ \text{V}_1}$$

3.2　ポリシ記述言語の実装

本節では，Datalog を基礎とした本言語の実装を説明する．Datalog は Prolog のサブセットとなる論理プログラミング言語で，本言語は宣言文と認可判定文を Datalog プログラムに翻訳することで実現する．本言語は，アクセス制御規則を論理プログラミングを用いて表現することにより，ポリシ記述に数理論理学的な裏付けを与えながら，宣言的な記述を行うことができる．

3.2.1　Datalog とその拡張実装 XSB の概要

Datalog は論理プログラミングを基礎とした言語で，特に大規模データベースと情報を交換するために設計されたルールベースの言語である．すなわち，データへ直接アクセスするインタフェースを用意してルールベースによる情報の交換をサポートする．また，Datalog は構文的観点から Prolog のサブセットであり，Datalog プログラムは Prolog インタプリタによって構文解析して実行することができる．

XSB[18] は Datalog の一実装で，ISO 標準に準拠した完全な Prolog システムとなっており，そのうえでさらに，テーブル型述語と非テーブル型述語の統合をサポートする．XSB は Prolog を含む通常の論理プログラミングシステムにはない以下のような特徴を持つ．

- SLG 導出[19] により整礎的意味論に従った完全な解導出が可能
- 高階論理プログラミング言語 HiLog[20] の実装
- Unification Factoring[21] をはじめとした様々な索引付け制御技術
- 移植性と拡張性のためのソースコード利用可能性

XSB のコンポーネントの多くは PSB-Prolog[22] に基づいているが，SLG 導出と HiLog 表現の処理のために Prolog システムと一部異なっている．たとえば，Prolog の SLD 導出[23] は深さ優先探索に基づくため無限ループに陥りやすいが，XSB の SLG 導出はほぼすべての論理プログラムを正しく評価できる．

3.2.2　Datalog プログラムへの変換規則

本言語では，3.1 節で示した構文規則に基づく各宣言文が，Datalog プログラムの節として変換され，認可判定文は，このプログラムに対する問合せとして評価する．Datalog に関係する用語は以下のように定義する．リテラル P は，述語の名前と一連の引数を持つ．引数は，定数項か変数項をとる．節は，$P_0 \leftarrow P_1, ..., P_n$ として書き，“←” の左辺にある 1 つの頭部リテラルと右辺にある本体リテラルのリストから構成される．

本実装では，構文規則で定義した各述語を以下のように Datalog リテラルへ変換する．“→” の左辺は，本言語による記述で，右辺は変換後の Datalog リテラルである．本変換では，宣言文と認可判定文の $fact$ 内にある述語を Datalog リテラルの述語とし，その目的語を Datalog リテラルの引数として変換する．また，**if** 以降にある $fact$ についても，上記手法によって Datalog リテラルした変換後，Datalog 節の条件節として用いる．

Policy specifies E **tagged** A → `tagged(E,A)`

Policy specifies A_1 **inherits** A_2 → `inherits(`A_1`,`A_2`)`

Policy specifies E_1 **tagged** A_1 **is permitted to** O E_2 **tagged** A_2
→ `permitted(`E_1`,`A_1`,O,`E_2`,`A_2`)`

Policy specifies E_1 **tagged** A_1 **is forbidden to** O E_2 **tagged** A_2
→ `forbidden(`E_1`,`A_1`,O,`E_2`,`A_2`)`

Policy specifies E_1 **tagged** A_1 **moves to** E_2 **tagged** A_2 **with** O
→ `moves(`E_1`,`A_1`,O,`E_2`,`A_2`)`

4.　アクセス制御モデルの表現手法

本章では，広く利用されている種々のアクセス制御モデルを本言語を用いて表現する手法を示す．はじめに，従来のように個々のアクセス制御規則を直接に記述するのではなく，変数やサブルーチンを用いたプログラムとして表現することで，要素間の関係や頻出する認可手順を簡潔に記述できることを説明し，その後，この表現手法を用いた具体的なアクセス制御規則の記述例を示す．

4.1　要素の階層化と認可手順の構造化

本節では，アクセス制御に利用する要素を階層化し，要素間の関係をサブルーチンとして構造化することで，ポリシ記述を簡略化する手法を示す．

要素の階層化は，様々なアクセス制御モデルに出現するポリシ記述の簡略化手法で，RBAC や DTE，Chinese Wall[24] などもアクセス制御要素をグループ化し，階層化する一方式として見ることができる．たとえば，RBAC は，アクセス制御規則を役割ごとに与えることでアクセス制御規則のグループ化を行い，さらに役割間の階層構造を定義することでアクセス制御規則群をまとめて継承する方法を定めるモデルである．同様に，DTE は，アクセス

主体とアクセス対象をそれぞれドメイン，タイプとしてグループ化し，その間にアクセス制御規則を定義することで柔軟にアクセス制御の粒度を制御するモデルである．

また，要素間の関係の構造化は，特定の認可手順をあらかじめサブルーチン化して簡潔に記述する手法で，たとえば，TBAC[7] や Ahmed らによる CSCW システム向けのアクセス制御モデル[25] は，それぞれ Authorization Step や Activity Template として特定の認可手順を抽象的に構造化している．これにより，サブルーチン化された認可手順をトランザクション処理などで繰り返し利用することで，ポリシの記述量を低減させ，見通しを向上させている．

本言語では，下記に示すように，アクセス主体とその属性を変数項に，それ以外の項を定数項として固定したうえで，条件節を用いてアクセス主体を指定することで，当該アクセス制御規則を適用するアクセス主体をまとめて指定することができる．

Policy specifies x **tagged** p **is permitted to** O_1 E_1 **tagged** A_1 **if** x **tagged** A_4 (1)

Policy specifies x **tagged** p **is permitted to** O_2 E_2 **tagged** A_2 **if** x **tagged** A_4 (2)

Policy specifies x **tagged** p **is permitted to** O_3 E_3 **tagged** A_3 **if** x **tagged** A_4 (3)

(1) は，属性 A_1 を付された対象 E_1 に操作内容 O_1 を許可する規則を，付された属性が A_4 であることを条件にアクセス主体 x に与えている．同様に (2)，(3) では，属性 A_2 を付された対象 E_2 に操作内容 O_2 を許可する規則と属性 A_3 を付された対象 E_3 に操作内容 O_3 を許可する規則を，付された属性が A_4 であることを条件にアクセス主体 x に与えている．これら (1)，(2)，(3) によって複数のアクセス制御規則を属性 A_4 にグループ化した後に，下記 (4) のように属性 A_4 を特定のアクセス主体 E_4 に付することにより，グループ化された (1)，(2)，(3) が改めて具体化されたアクセス主体に (5)，(6)，(7) として展開される．

Policy specifies E_4 **tagged** A_4 (4)

Policy specifies E_4 **tagged** A_4 **is permitted to** O_1 E_1 **tagged** A_1 (5)

Policy specifies E_4 **tagged** A_4 **is permitted to** O_2 E_2 **tagged** A_2 (6)

Policy specifies E_4 **tagged** A_4 **is permitted to** O_3 E_3 **tagged** A_3 (7)

このとき，(5)，(6)，(7) は，直接記述された規則ではなく，グループ化された規則を付与，展開することで新たに生成された規則であり，(4) のように具体的なアクセス主体に属性 A_4 を付すことで新たな規則を繰り返し生成することが可能である．

加えて，本言語では上記のようにグループ化された属性間の遷移条件を述語 “**moves to**” を用いて下記のように定めることにより，特定の認可手順を抽象的に構造化する．

Policy specifies x **tagged** p **moves to** x **tagged** A_5 **with** O_3 **if** x **tagged** A_4 (8)

Policy specifies x **tagged** A_5 **is permitted to** O_4 E_5 **tagged** A_6 (9)

(8) は，アクセス主体 x に属性 A_4 が付されていたとき，O_3 を契機にその属性が A_5 に遷移する規則を示す．この遷移により，アクセス主体 x の属性は，属性 A_5 に変わる．(9) は，属性 A_5 を付されたアクセス主体に，属性 A_6 を付されたアクセス対象 E_5 に対して O_4 を許可する規則を定める．これらにより，A_4 によってグループ化されたアクセス規則が，さらに (8) によって (9) に遷移することで，(1)，(2)，(3) から (9) へ続く認可手順を抽象的に構造化している．また，遷移規則は複数連続させることでトランザクション処理のような複雑な認可手順も簡潔に表現できる．

4.2　アクセス制御規則の記述例

本節では，前節で示した記述手法を用いて，広く利用されている種々のアクセス制御モデルを実際に表現する記述例を示す．ここでは階層化の例として RBAC を，認可手順のサブルーチン化の例として TBAC を取り上げる．

RBAC　RBAC は，役割として抽象化された属性に複数のアクセス制御規則を関連付けることでポリシを記述するモデルで，企業などの組織形態に自然に適用しやすいため，広く一般的に利用されている．下記の例では，(10) によって Manager の役割を付されたアクセス主体に Billing_Information を付されたアクセス対象を Read する権限を与え，(11) によって Floor_Leader の役割を付されたアクセス主体に Accounting_Information を付されたアクセス対象を Read する権限を与えている．また，(12) によって，Manager は，Floor_Leader に与えられた権限を継承する．これらの規則により，仮に同じアクセス主体であっても付された役割によって与えられる権限を変更することができる．

Policy specifies x **tagged** p **is permitted to** Read y **tagged** q **if** x **tagged** Manager, y **tagged** Billing_Information (10)

Policy specifies x **tagged** p **is permitted to** Read y **tagged** q **if** x **tagged** Floor_Leader, y **tagged** Accounting_Information (11)

Policy specifies Manager **inherits** Floor_Leader　(12)

TBAC　TBAC は，Authorization Step（AS）として認可手順をまとめたうえで，アクセス主体，アクセス対象，操作内容に加えて，AS 名と AS 内のどの工程にいるか，という情報を指定して認可判定を決定するモデルである．本モデルでは，トランザクション処理中の各要素について，多様な関係を指定することができる．下記の例では，(13) によって Prepare_Phase にあるアクセス主体に，Not_Prepared にある DB1 に対して Request_to_Prepare する権限を与え，(14) によって Commit_Phase にあるアクセス主体に，Prepared にある DB1 を Commit する権限を与えている．また，(15) によって，Prepare_Phase にあるアクセス主体は，DB1_Prepared を契機に Commit_Phase に遷移するように指定している．これらの規則により，トランザクション処理の工程に応じて与えられる権限を変更することができる．

Policy specifies x **tagged** p **is permitted to** Request_to_Prepare DB1 **tagged** q **if** x **tagged** Prepare_Phase, DB1 **tagged** Not_Prepared　(13)

Policy specifies x **tagged** p **is permitted to** Commit DB1 **tagged** q **if** x **tagged** Commit_Phase, DB1 **tagged** Prepared　(14)

Policy specifies x **tagged** Prepare_Phase **moves to** x **tagged** Commit_Phase **with** DB1_Prepared　(15)

5. ポリシ記述言語の評価

本章では，本言語の有用性を評価するために行った実験の結果とその考察を示す．はじめに，本言語の妥当性を確認するために，SELinux で利用されているポリシを本言語を用いて実際に構成し，認可判定時の応答内容の同一性を確認することで，本言語が論理的に正しく動作していることを実証した．その後，本言語の表現力を評価するために，同内容のポリシを構成するために必要な記述量を SELinux のポリシ記述手法と本言語で複数の観点から定量的に比較し，本言語の表現手法が実際のポリシを簡潔に表現できることを確認した．最後に，これらの結果をふまえて本言語の利害得失について考察する．

5.1 実験システムの構成

実験システムは，Pentium4（3.0 GHz），メモリ 1 GB の計算機上に実装したもので，OS 環境は，Debian/GNU Linux に SELinux 関連のパッケージを追加したものを利用する．主要なソフトウェアのバージョンは，libselinux1(2.0.65-5)，selinux-basics(0.3.5)，selinux-policy-default(2:0.0.20080702-6) である．実験システムでは，カーネル空間に存在する sidtab と avtab のアドレスを取得し，そのハッシュ配列のポインタを利用してテーブルの内容をすべて取得する．同時に，取得した avtab に並んでいる全要素の情報を本言語によって表現したうえで，Datalog に読み込ませた認可判定機構を用意し，ソースコンテキスト，ターゲットコンテキスト，セキュリティクラスの組合せによる問合せに備える．計測プログラムは C 言語で記述し，結果はコンテキストやクラスの情報を名前の文字列に変換し，テキストファイルとして記録した．

5.2 応答内容の比較による妥当性の評価実験

本節では，SELinux のポリシと，同ポリシを本言語で構成したもので，応答内容の同一性を評価した結果を示す．実験は，sidtab の全要素を用いて，セキュリティクラスに対する総当りで *context_struct_compute_av*() を呼び出して AV（Access Vector）を取得したうえで，同じソースコンテキスト，ターゲットコンテキスト，セキュリティクラスの組合せで Datalog に問合せを行い，得られた AV が *context_struct_compute_av*() が返したものと一致するか否かを評価した．

この実験では，sidtab の全要素と全セキュリティクラスの組合せを用いた合計 15,100,162 個の問合せが生成されて評価され，そのうち，双方が同じ AV によって応答したものが 875,644 個，何の AV も返さないという意味で応答内容が一致したものが 14,216,639 個あった．結果として，合わせて 99%以上，15,092,283 個の問合せにおいて，応答内容が SELinux と本言語で同一であった．また，残りの約 0.05%，7,879 個の問合せは応答内容が一致していなかった．

5.3 記述量の比較による表現力の評価実験

本節では，SELinux のポリシを本言語で表現する場合に必要となる記述量を，ポリシを構成するために必要となるソースコードの行数とリスティングした場合のページ数によって比較した結果を示す（**表 1**）．

本言語においては 4 章で示した表現手法を用いることで，同内容のアクセス制御規則であっても多様な表現を許容するため，本実験では特に 2 種類の方式で記述を行った．1 つ目は，ケイパビリティリスト（C-List）[26] のようにアクセス主体の属性としてのソースコンテキストを定数項として指定し，アクセス対象の属性としてのターゲットコンテキストとセキュリティクラス，操作内容としての AV を変数項として指定したうえで，条件節内でそれら変数項の内容を具体的に特定する記述方式で，2 つ目は，アクセス制御リスト（ACL）[27]

表 1　ソースコード行数とリスティングページ数の比較
Table 1　SLOC comparison of policy expressions.

評価対象＼評価指標	ソースコード行数	リスティングページ数
SELinux ポリシソース	6,524	133
提案言語記述方式（C-List）	335	95
提案言語記述方式（ACL）	356	96

のようにアクセス対象の属性としてのターゲットコンテキストを定数項として指定し，アクセス主体の属性としてのソースコンテキストとセキュリティクラス，操作内容としての AV を変数項として指定したうえで，条件節内でそれら変数項の内容を具体的に特定する記述方式である．

この実験では，SELinux の記述手法によってソースコード行数 6,524 行で記述されたポリシが，本言語を利用した C-List 式の記述方式では 335 行，ACL 式の記述方式では 356 行で記述された．また，リスティングしたページ数としては，SELinux ポリシのソースファイルで 133 ページあったものが，本言語で記述した場合に C-List 式，ACL 式それぞれで 95 ページ，96 ページになった．

5.4　実験結果の考察

妥当性の評価実験では，アクセス可として判定された問合せとアクセス不可として判定された問合せで計 99%以上の応答内容が一致したが，一部応答内容が一致しないケースが確認された．この原因は，システム運用時に動的に生成されるコンテキストに対して問合せを行ったことによるもので，これについては認可判定時の引数の渡し方に依存するものである．そのため，この解決は認可判定機構の外部で工夫する必要があるが，一方で，元々の SELinux の認可判定がポリシの記述に対して妥当であることを前提としたとき，本言語によるポリシ記述とそれに基づいた認可判定自体も妥当であったといえる．したがって，この実験結果により，論理プログラミングを基礎とした本言語が，現実のポリシを論理的に正しく記述することができることを示した．

この応用としては，論理プログラミングの持つ特徴をアクセス制御に適用することで，たとえば，変数を用いた問合せによりポリシの検証を柔軟に行い，その検証結果をアクセス制御の条件として記述すること，メタな規則によって新規則の追加方法や既存規則の変更を制御すること，認可判定に利用する引数を認可判定機構が状況に応じて変更することなどが考えられる．

表現力の評価実験では，本言語による記述方式により SELinux のポリシを少ない記述量で構成できることを示したが，これは，アクセス制御規則を定めるのに必要な多くの指定を条件節にまとめたからであり，結果として，各規則に対する条件節の内容が多くなっている．そのため，必要な要素を直接指定している SELinux の記述手法の方が，各々の規則を個別に見た場合には内容を直感的に把握しやすい．一方で，本言語の記述手法では，アクセス制御規則を階層化やサブルーチン化によってある程度まとめて記述するため，各々の規則を見た場合の可読性はそれほどよくないが，ポリシ全体としての見通しは向上している．

したがって，SELinux のポリシ記述手法のような直接要素を指定する表現と，本言語のような変数項と条件節を用いた表現は，双方に異なる特徴があるため，利害得失は利用フェーズに依存すると考えられる．たとえば，他の規則と関連性を持たないような少数のアクセス制御規則を厳密に記述していきたいような場合は前者の記述手法が適しているし，様々な認可判定で共有されるアクセス手順が多く存在し，それをまとめて指定したい場合や，アクセス制御規則の総数が膨大になるため，柔軟な構造化が必要な場合には後者の方が適している．

6. ま と め

本稿では，論理プログラムとしてアクセス制御規則を表現することで，属性の継承やサブルーチンとして構造化された記述が可能なポリシ記述言語を提案し，多層防御を効果的に適用する際に課題となるポリシ記述に係る複雑性を解決する手法を示した．本言語は，個々のアクセス制御規則を記述するための宣言文と，宣言文の集合として構成されたポリシに問合せを行うための認可判定文を，構文規則とその形式的な意味，推論規則を定めることで基本設計を行い，これを Prolog のサブセットである Datalog を用いて実装している．また，本稿では，本言語を用いて具体的なアクセス制御モデルを構造的に記述する手法を示したうえで，SELinux のポリシを実際に記述した実験システムを構成し，認可判定の妥当性と表現力の評価を行った．妥当性の評価実験では，SELinux の認可判定が妥当であることを前提としたときに，本言語による認可判定が論理的に正しい応答を示し，妥当であることを実証した．さらに，表現力の評価実験では，本言語を用いた記述手法が SELinux のポリシを少ない記述量で構成できることを示し，両実験結果から本言語の有用性について考察を行った．

今後は，まず，2.3 節で期待される効果として示したような本言語の応用について，実際に実現する方式について検討する．また，本言語を用いて実際のシステムのアクセス制御を行うような認可判定機構を構成し，OS などのアクセス制御機構と連携させる方式について

も検討する．その際には，認可判定機構そのものをセキュアに構成するため，また，性能を向上させるという観点からも，Datalog の処理系をカーネルに埋め込むことを計画している．

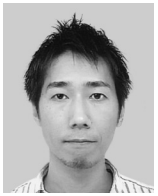
謝辞　貴重なご助言をいただいた編集委員と匿名査読者の方々に感謝いたします．

参 考 文 献

1) Bass, T. and Robichaux, R.: Defense-in-depth revisited: Qualitative risk analysis methodology for complex network-centric operations, *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, Vol.1, pp.64–70 (2001).
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*, Berkeley, CA, USA, USENIX Association, pp.29–42 (2001).
3) Warren, D.H.D., Pereira, L.M. and Pereira, F.: Prolog – the language and its implementation compared with Lisp, *Proc. 1977 Symposium on Artificial Intelligence and Programming Languages*, New York, NY, USA, ACM, pp.109–115 (1977).
4) Ceri, S., Gottlob, G. and Tanca, L.: What you always wanted to know about Datalog (and never dared to ask), *IEEE Trans. Knowledge and Data Engineering*, Vol.1, No.1, pp.146–166 (1989).
5) Ferraiolo, D.F. and Kuhn, D.R.: Role-Based Access Control, *15th National Computer Security Conference*, Baltimore, MD, pp.13–16 (1992).
6) Tidswell, J. and Potter, J.: An approach to dynamic domain and type enforcement, Lecture Notes in Computer Science, Vol.1270, pp.26–37 (1997).
7) Banerjee, A. and Naumann, D.: History-based access control and secure information flow, Lecture Notes in Computer Science, Vol.3362, pp.27–48 (2005).
8) Spencer, R., Corporation, S.C., Smalley, S., Loscocco, P., Agency, N.S. and Andersen, M.H.D.: The Flask Security Architecture: System Support for Diverse Security Policies, *Proc. 8th USENIX Security Symposium*, pp.123–139 (1999).
9) KaiGai, K.: Security Enhanced PostgreSQL (2006). http://code.google.com/p/sepgsql/
10) Macmillan, K., Brindle, J., Mayer, F., Caplan, D. and Tang, J.: Design and implementation of the SELinux policy management server, *Proc. Security Enhanced Linux Symposium*, pp.1–6 (2006).
11) Abadi, M.: On SDSI's linked local name spaces, *J. Comput. Secur.*, Vol.6, No.1-2, pp.3–21 (1998).
12) Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B. and Ylonen, T.: *SPKI Certificate Theory*, chapter 2693, RFC Editor (1999).
13) Blaze, M., Feigenbaum, J. and Strauss, M.: Compliance Checking in the PolicyMaker Trust Management System, *FC '98: Proc. 2nd International Conference on Financial Cryptography*, London, UK, pp.254–274, Springer-Verlag (1998).
14) Blaze, M., Feigenbaum, J. and Keromytis, A.D.: KeyNote: Trust Management for Public-Key Infrastructures (Position Paper), *Proc. 6th International Workshop on Security Protocols*, London, UK, pp.59–63, Springer-Verlag (1999).
15) Becker, M., Fournet, C. and Gordon, A.: Design and Semantics of a Decentralized Authorization Language, *CSF '07: Proc. 20th IEEE Computer Security Foundations Symposium*, Washington, DC, USA, IEEE Computer Society, pp.3–15 (2007).
16) Halpern, J.Y. and Weissman, V.: Using First-Order Logic to Reason about Policies, *ACM Trans. Inf. Syst. Secur.*, Vol.11, No.4, pp.1–41 (2008).
17) OASIS: eXtensible Access Control Markup Language (XACML) Version 2.0 core specification (2005). http://www.oasis-open.org/committees/xacml/
18) Rao, P., Sagonas, K.F., Swift, T., Warren, D.S. and Freire, J.: XSB: A System for Effciently Computing WFS, *LPNMR '97: Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, London, UK, pp.431–441, Springer-Verlag (1997).
19) Chen, W. and Warren, D.S.: Tabled evaluation with delaying for general logic programs, *J. ACM*, Vol.43, No.1, pp.20–74 (1996).
20) Chen, W., Kifer, M. and Warren, D.S.: HiLog as a platform for database languages, *Proc. 2nd international workshop on Database programming languages*, San Francisco, CA, USA, pp.315–329, Morgan Kaufmann Publishers Inc. (1989).
21) Dawson, S., Ramakrishnan, C.R., Ramakrishnan, I.V., Sagonas, K., Skiena, S., Swift, T. and Warren, D.S.: Unification factoring for efficient execution of logic programs, *POPL '95: Proc. 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, NY, USA, ACM, pp.247–258 (1995).
22) Wielemaker, J., Huang, Z. and Van der meij, L.: Swi-prolog and the web, *Theory Pract. Log. Program.*, Vol.8, No.3, pp.363–392 (2008).
23) Apt, K.R.: Introduction to Logic Programming, Technical report, Austin, TX, USA (1988).
24) Brewer, D. and Nash, M.: The Chinese Wall security policy, *Proc. IEEE Symposium on Security and Privacy*, pp.206–214 (1989).
25) Ahmed, T. and Tripathi, A.R.: Specification and verification of security requirements in a programming model for decentralized CSCW systems, *ACM Trans. Inf. Syst. Secur.*, Vol.10, No.2, p.7 (2007).
26) Fabry, R.S.: Capability-based addressing, *Comm. ACM*, Vol.17, No.7, pp.403–412 (1974).
27) Saltzer, J. and Schroeder, M.: The protection of information in computer systems,

Proc. IEEE, Vol.63, No.9, pp.1278–1308 (1975).

(平成 21 年 11 月 30 日受付)
(平成 22 年 6 月 3 日採録)



橋本　正樹（正会員）

2001 年立命館大学文学部人文総合科学インスティテュート卒業．同在学中に有限会社ワイプ設立・取締役情報システム担当就任．2010 年情報セキュリティ大学院大学情報セキュリティ研究科修了，博士（情報学）．2010 年 4 月より情報セキュリティ大学院大学情報セキュリティ研究科助教．同研究科にて，主に OS によるアクセス制御の研究と教育に従事．電子情報通信学会，IEEE 各会員．電子情報通信学会 ISS・情報通信システムセキュリティ研究会専門委員．

金　　美羅（正会員）

1996 年釜慶大学自然科学部電子計算学科卒業．1998 年同大学院修士課程修了．2003 年東京大学大学院情報理工学系研究科電子情報学専攻博士課程修了，博士（情報理工学）．同年東京大学生産技術研究所研究員．2004 年情報セキュリティ大学院大学助教．セキュアシステムの研究開発に従事．2009 年情報セキュリティ大学院大学客員研究員．電子情報通信学会会員．

辻　　秀典（正会員）

1996 年東京工業大学工学部情報工学科卒業．2001 年東京大学大学院工学系研究科情報工学専攻修了，博士（工学）．株式会社インターネット総合研究所を経て，2002 年株式会社情報技研を設立．同代表取締役社長．2004 年情報セキュリティ大学院大学客員准教授兼任，現在に至る．IEEE, ACM，USENIX 各会員．電子情報通信学会 HCG，料理メディア研究会専門委員．主な著書は『できる PRO Linux/Fedora』シリーズ（インプレスジャパン）．

田中　英彦（名誉会員）

1970 年東京大学大学院工学系研究科電気工学専門課程修了，工学博士．東京大学にて計算機アーキテクチャ，並列処理，人工知能，自然言語処理，分散処理，メディア処理等の教育・研究に従事．東京大学工学部教授，同大学院情報理工学系研究科長を経て，2004 年情報セキュリティ大学院大学情報セキュリティ研究科長・教授．情報処理学会名誉員，人工知能学会論文賞，ACM SIGGRAPH'99 Impact Paper Award，人工知能学会功績賞，東京都科学技術功労者表彰，経済産業大臣表彰等受賞．情報・システム研究機構教育研究評議会評議員，日本学術会議会員，IEEE Fellow，東京大学名誉教授．