社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DPS－122（66）
2005－CSEC－28（66）
2005／3／23

機能マッピングによるセキュリティポリシーの矛盾解消

松田 勝志, 岡城 純孝, 小川 隆一

NEC インターネットシステム研究所

概要

トップダウンに策定されたセキュリティポリシーによるセキュリティ対策は上手くいかない場合がある. そのようなセキュリティポリシーは実際に適用するネットワークシステムの構成や利用形態と乖離していることがあるためである. 本稿では, セキュリティポリシーとネットワークシステムの構成を対応付けることでセキュリティポリシーの矛盾を検出し, 正しいセキュリティポリシーに修正することを支援する方式について述べる. ポリシーの矛盾には, ポリシーと構成の対応関係に起因する表層的な矛盾とセキュリティ的な運用知識に起因する意味的な矛盾があり, 提案方式はこれらの矛盾を検出できることを示す.

Conflict Resolution of the Security Policy by Functional Mapping

Katsushi MATSUDA, Sumitaka OKAJO and Ryuichi OGAWA

Internet Systems Research Laboratories, NEC Corp.

Abstract

Security management with a security policy sometimes do not work, because there is a great difference between the security policy and the topology or usage manners of the network system to which the policy will be applied. In this paper, we describe a method that discovers inconsistencies in the policy by comparing a fragment of the policy with a component of the network system and supports to amend the policy into an exact one. The method can detect two kinds of inconsistencies, namely, a syntactic inconsistency caused by the mapped form of a policy and a topology and a semantic inconsistency caused by the operating knowledge in the security field.

1 はじめに

セキュリティポリシーを策定し, ファイアウォールやアクセス制御によるネットワークシステムのセキュリティ対策が行われているにも関わらず, 不正アクセスやワーム感染や情報漏えい等の脅威は一向に低下しない. この原因の一つに, 実際に運用されるネットワークシステムの構成や利用形態から乖離したセキュリティポリシーが適用されることが考えられる.

そこで本稿では, 策定したセキュリティポリシーと実際に適用するネットワークシステムの構成とを対応付ける機能マッピング方式と, それら対応関係を元にセキュリティポリシーの矛盾を検出･解消するシステムについて述べる.

2 セキュリティポリシーの矛盾

企業がセキュリティに取り組む場合, セキュリティポリシーを策定し, それに基づいたセキュリティ対策を実施することが多い[1][2]. 図１にセキュリティポリシーの階層モデルを示す.

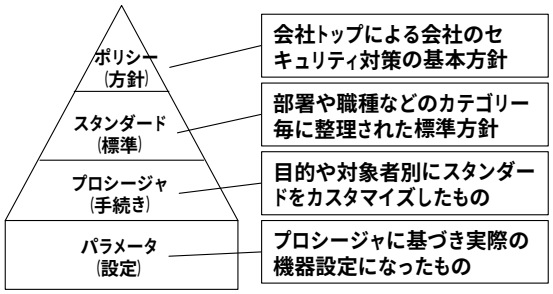

図１．ポリシー階層モデル

本モデルに基づいた場合, セキュリティポリシーの策定は，最も抽象的な「方針」から作成し, 部門や目的に応じて順に具体化することで「標準」「手続き」を作成する．そして最後に現場のセキュリティ管理者やシステム管理者が「手続き」に基づいて機器の「設定」を作成・適用することで対策となる. 本稿で扱うセキュリティポリシーとは, 図 1 の「標準」および「手続き」である.

上述のようにセキュリティポリシーはトップダウンに詳細化してから実際の対象ネットワークシステムに適用される．そのため, 策定したセキュリティポリシーが実際の対象ネットワークシステムの環境に合わない場合や, 想定外のネットワークシステムである場合がある. このような不完全なセキュリティポリシーを実環境に適用すると, セキュリティホールができるなどの大きな問題が生じる場合がある. これをセキュリティポリシーの矛盾と呼ぶ.

筆者らは, この問題を解決するための機能マッピング方式を開発している[3]．本稿では, セキュリティポリシーの表面的な矛盾だけではなく, 意味的な矛盾まで検出することができる機能マッピング方式と矛盾検出方式の拡張について述べる.

3. 機能マッピング

機能マッピングとは, セキュリティポリシーの個々の宣言と実際に適用する対象ネットワークシステムの機器またはソフトウェアとをセキュリティ機能によって対応付ける方式である．本節では，機能マッピングで用いる記述方式と知識, そして機能マッピングのアルゴリズムについて述べる.

3.1 機能クラス

あるセキュリティポリシーにおいて記述された機能と別のセキュリティポリシーで記述された機能が, 同一であるか, 全く別のものか, 関連するものか, を識別できなければならない. これらを識別するために, 機能クラスというセキュリティ機能を表現する共通語彙体系を導入した. 機能クラスは, セキュリティ機能に特化した抽象クラスであり, ツリー構造で階層的に分類したものである. また, 機能クラスには, セグメントの種類も含んでいる. 図 2 に機能クラスの例を示す.

```
network.segment.dmz                 DMZセグメント
network.segment.internet            インターネットセグメント＝インターネット
network.segment-boundary.dmz-int    DMZとインターネットとのセグメント境界
network.segment-boundary.intra-wan  イントラネットとWANとのセグメント境界
function.router                     ルーティング機能
function.filtering.packet           パケットフィルタリング機能
function.filtering.contents         コンテンツフィルタリング機能
function.address.ipmasqarade       IPマスカレード機能
function.service.integrity          ファイル改竄防止サービス機能
function.service.job                バックグランドジョブサービス機能
function.encode.des.54              54ビットDES暗号方法機能
account.generic.root                rootアカウント
account.generic.user                一般ユーザアカウント
file.acl.readonly                   読み込み可能のファイルアクセス権
file.acl.readwrite                  読み書き可能のファイルアクセス権
file.password                       パスワードファイル
packet.forward                      パケット転送
packet.ip.tcp.httpd                 80/TCPパケット
packet.ip.udp.1434                  1434/UDPパケット
information.fingerprint             フィンガープリント情報
....
```

図 2．機能クラスの例

機能クラスは, ドット(.)がツリー構造のアークであり, ノードを重ねることにより詳細なセキュリティ機能を表現するように構成されている．例えば，function.filtering.packet という機能クラスは, パケットフィルタリング機能のことであり，function.filtering.contents という機能クラスはコンテンツフィルタリング機能のことである．更に，これらの機能クラスは function.filtering というフィルタリング機能に属するため, セキュリティ的に近い機能であることが分かる.

3.2 セキュリティポリシー記述

セキュリティポリシーを機能クラスを用い, セグメント別に記述できるようにしたものがセキュリティポリシー記述である.

機能マッピングで用いる記述方式では, セグメントという概念を取り入れている. セグメントとは, ある目的のために明確に分離されたネットワークのゾーン, およびそれらゾーン間の境界部分である．セグメントの例としては，DMZ セグメントや LAN セグメント，そして DMZ-INT セグメント（DMZ とインターネットの境界セグメント）等がある．これらのセグメント毎にセキュリティポリシーを分けるこ

とで目的別に分類することが可能となる.

図 3 にセキュリティポリシー記述の例を示す. 一般にセキュリティポリシーとは, 複数の宣言によって構成されており, 個々の宣言もセキュリティポリシーと呼ばれることも少なくない. 本稿では厳密に区別するために, 個々の宣言をルールと呼び, ルールの集合をセキュリティポリシーと呼ぶ.

```
<standard name="DMZ-001-1">
        <comment>ログを取る</comment>
        <subject>function.logging</subject>
        <enforce>add</enforce>
        <segment>network.segment.dmz</segment>
</standard>

<standard name="DMZ-002-1">
        <comment>ネットワークサービスの管理をする</comment>
        <subject>function.authorize.service</subject>
        <enforce>add</enforce>
        <segment>network.segment.dmz</segment>
</standard>
<procedure name="DMZ-002-1-1">
        <comment>必要のないネットワークサービスを起動しない</comment>
        <parent>DMZ-002-1</parent>
        <subject>service.*</subject>
        <set>disauthorize</set>
</procedure>
```

図 3. ポリシーの例

ルールは, あるセキュリティ機能をあるセグメントにおいて実現するか否かを表す standard ルールと, standard ルールで指定されたセキュリティ機能をどう実現するかを表す procedure ルールからなる. 例えば, 最初の DMZ-001-1 は, DMZ セグメントにおいて, ログ(function.logging)を取る(add), というルールである. このセキュリティポリシー記述は, あるルールがどのセキュリティ機器で実施すべきか書かれていない. すなわち, あるセグメントにおいて実現すべきセキュリティ機能は記述するが, どのようなネットワーク機器やソフトウェアでそのセキュリティ機能を実現するかは記述しておらず, 実際の文書によって策定されたセキュリティポリシーと同じ記述内容となっている.

なお, 本記述言語は, セキュリティポリシー言語 SCCML[4]の上位言語にあたり, セキュリティ機器のより具体的な設定の記述は SCCML を用いる.

3.3 トポロジー記述

対象ネットワークシステムの構成を記述するのが, トポロジー記述である. トポロジー記述は, 対象ネットワークシステムのセグメント構成, ハードウェア構成, そしてハードウェア上のソフトウェア構成をそれぞれ関係付けて記述することができる. 本稿では, ハードウェアとソフトウェアを総称してノードと呼ぶ. 図 4 にトポロジー記述を用いて記述したトポロジーの例を示す.

```
<topology>
    <segment name="int" type="network.segment.internet">
        <address>0/0</address>
    </segment>
    <segment name="int-dmz" type="network.segment-boundary.int-dmz">
        <address>207.100.5.233</address>
        <address>207.190.1.1</address>
        <hardware name="fw0">
            <canonical>NEC Express5800</canonical>
            <model>54wd</model>
            <nic name="eth0">207.100.5.233</nic>
            <nic name="eth1">207.190.1.1</nic>
            <software name="fw0-os">….</software>
            <software name="pf1s">
                <canonical>iptables</canonical>
                <version>1.4.3</version>
            </software>
            ・・・
    <segment name="dmz" type="network.segment.dmz">
        <address>207.190.1.1/24</address>
        <hardware name="www">
            ・・・
</topology>
```

図 4. トポロジーの例

トポロジー記述は, ポリシー記述とは反対に, あるノードがどのルールを実現できるか書かれていない. すなわち, セキュリティポリシーからは独立した記述となっている.

3.4 ノード知識

ノード知識は, ネットワークシステムのノード, すなわち個々のハードウェアやソフトウェアが, セキュリティ上のどのような機能を有しているのかを前述の機能クラスを用いて列挙記述した知識ベースである. 図 5 にノード知識の例を示す.

```
<software canonical="ipchains" version="1.3.10">
        <function>function.router</function>
        <function>function.filtering.packet</function>
        <function>function.address.ipmasquerade</function>
</software>
<software canonical="iptables" version="1.2.7">
        <inherit canonical="ipchains" version="1.3.10"/>
        <function>function.address.snat</function>
        <function>function.address.dnat</function>
</software>
```

図 5. ノード知識の例

図 5 の例では, パケットフィルタリングソフトウェアである ipchains と netfilter(iptables)のセキュリティ機能を機能クラスを用いて表現している. ipchains は, ルーティングの機能(function.router), パケットフィルタリング

の機能(function.filtering.packet)とアドレス変換の機能(function.address.ipmasquerade)を持っていることがノード知識から分かる.

3.5 機能マッピングのアルゴリズム

以上の記述および知識を用いて機能マッピングは処理を行う. 図 6 に機能マッピングのアルゴリズムを示す.

```
begin
    for Ri ∈ Policy do
        begin
            s := segment(Ri);
            M := {};
            for Ni ∈ Nodes do
                if segment(Ni) = s then begin
                    Fs := functions(Ni);
                    f := function(Ri);
                    if f ∈ Fs then
                        M := M + tuple(Ri,n);
                end
        end
end;
```

図 6. 機能マッピングのアルゴリズム

segment 関数は, 引数がルールの場合はルールのセグメントを返し, 引数がノードの場合はノードの属するセグメントを返す. functions 関数はノード知識を参照して, 引数で指定されたノードの持つセキュリティ機能のリストを返す. function 関数は引数で指定されたルールで宣言されているセキュリティ機能を返す. tuple 関数はルールとノードからなるタプルを作成する.

この機能マッピングアルゴリズムによって, 入力されたポリシー記述とトポロジー記述における全てのタプル(ルールとノードの対応関係)を導出することができる.

4 矛盾検出と解消

一般にセキュリティポリシーは, ポリシー中のルールの全てが対象ネットワークシステムのトポロジー中のノードのいずれかに対応付けられ(ルールで指定されているセキュリティ機能を実現するノードが存在する), 且つ対象ネットワークシステムのノードの全てがポリシー中のルールのいずれかに対応付けられて(ノードのセキュリティ機能を運用するルールが存在する)いることが望ましい.

しかしながら, 実際には, 無駄なルールやノードがある場合や, ポリシー策定において誤って矛盾するルールを挿入してしまう場合がある. 正しいセキュリティ運用のためには, これらの矛盾や不整合を検出し, 修正しなければならない.

機能マッピングが導出した全てのタプルの種類や組み合わせを調べることで上述の矛盾や不整合を検出することができる. この矛盾検出には表層的矛盾検出と意味的矛盾検出がある. 以降これらの矛盾検出とその解消方法について述べる.

4.1 表層的矛盾の検出

表層的矛盾とは, 機能マッピングの結果のタプルおよび記述のシンボルのみを利用して判定できる表面的な矛盾である. ルールで宣言されたセキュリティ機能の意味やセキュリティ的またはシステム的な決め事やノウハウは一切利用しない. このような矛盾は, タプルの形式のみで検出することができる. 矛盾か否かを判定する矛盾判定式を表 1 に示す(矛盾以外の全ての関係を列挙している).

番号	機能とノードの関係	矛盾
(1)	∃n(fa $ n ∧ ¬(fb $ n ∧ fa≠fb))	-
(2)	∃n(¬fa $ n ∧ ¬(¬fb $ n ∧ fa≠fb))	-
(3)	∀n(¬ ∃f(f $ n))	overpolicy
(4)	∀n(¬ ∃f(¬f $ n))	-
(5)	∀f(¬ ∃n(f $ n))	underpolicy
(6)	∃n(fa $ n ∧ fb $ n ∧ ...)	-
(7)	∃n(¬fa $ n ∧ ¬fb $ n ∧ ...)	-
(8)	∃n(fa $ n ∧ ¬fb $ n ∧ ... ∧ fa≠fb ∧ ...)	-
(9)	∃n(fa $ n ∧ ¬fa $ n)	collision
(10)	∃f(f $ n1 ∧ f $ n2 ∧ ...)	-
(11)	∃f(¬f $ n1 ∧ ¬f $ n2 ∧ ...)	-
(12)	∃fa∃fb(fa $ n1 ∧ fb $ n2 ∧fa⊥fb)	collision
(13)	∃fa∃fb(¬fa $ n1 ∧ ¬fb $ n2 ∧ ¬fa⊥¬fb))	collision
(14)	∃f∃h(f $ n1 ∧ f $ n2 ∧ ... ∧ n1,n2,..∈h)	collision

表 1. 矛盾判定式

f はルールで定義されたセキュリティ機能を実現する, ¬f はそのセキュリティ機能を実現しない, をそれぞれ表す. n はノード, h はあるハードウェアを表している. また, 記号$は機能マッピングによるタプルを, 記号⊥はルールに指定されている互いに独立なセキュリティ機能をそれぞれ示している. 例えば, 表 1 の(3)は, あるノードがいずれのルールともタプルとなっていないことを表している.

これらの判定式によって検出する矛盾には,

Collision(ポリシー衝突)と Overpolicy(ポリシー過剰)と Underpolicy(ポリシー過少)がある(表 1 の第 3 列,「-」は矛盾なしを表す).

Collision とは，お互いに相反する一つ以上のルールが, ある一つのノードに対応付けられた場合を言う．典型的な例は，表 1 の(9)であり, 相反する機能が同じノードに対応付けられている場合である.

Overpolicy とは，ルールによって実現を規定しているセキュリティ機能があるにも関わらず, それを実現するノードが存在しない場合である(表 1 の(3)).

Underpolicy とは, どのルールとも対応付けられなかったノードが存在する場合である. すなわち, セキュリティ機能を持ったノードがインストールされているにも関わらず, そのノードの機能をどうするか(実施するかしないか)がルールとして定義されていない場合である(表 1 の(5)).

4.2 表層的矛盾の解消

前節の矛盾判定式によって, 表層的矛盾が検出できる. これらの矛盾を解消する方法について示す.

Collision には，大きく分けて 3 種類の競合状態がある.

(A) 正反対のルール
(B) 同じノードに同時に実現不可と規定されたルール
(C) 同じハードウェア上では実現不可と規定されたルール

このうち，(A)と(B)の Collision は，1 つのノードに対応するルールが複数という形であるため, いずれか一方のルールを削除することで矛盾が解消できる．(C)の Collision は，同一ハードウェア上で複数のノードに対応付けられないように, ノードを削除することで矛盾が解消できる.

Overpolicy は，ルールはあるが，そのルールのセキュリティ機能を実現するノードが存在しない状態であるため, この矛盾を解消するには，ルールを削除するか，ルールのセキュリティ機能を実現するノードを追加する.

Underpolicy は，ノードはあるが，そのノードが持つセキュリティ機能を用いるルールが存在しない状態であるため, この矛盾を解消するには，ノードを削除するか，そのノードのセキュリティ機能を用いるルールを追加する.

4.3 表層的矛盾検出と解消の動作例

機能マッピングと矛盾検出を行うシステムの実装を行った. 図 7 に機能マッピングの結果と表層的矛盾検出の例を示す.

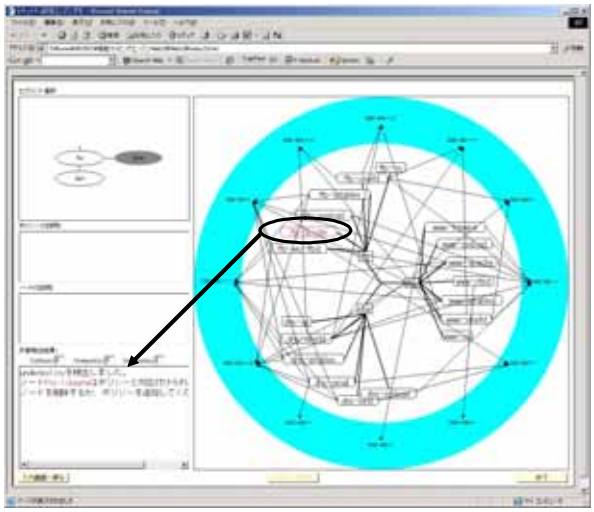

図 7. 表層的矛盾の検出

システムはセグメント毎にポリシーとノードの対応関係をグラフで表現する(外周部にポリシー，内部にノードが配置されており，対応は線分で表されている)．図 7 の画面ダンプでは，表層的矛盾検出によって，Underpolicy が存在することを示している.

4.4 意味的矛盾の検出

意味的矛盾とは, ルールで宣言されたセキュリティ機能の意味や, セキュリティ的またはシステム的な運用知識に基づいて判定できる意味的な矛盾である．例えば，運用知識として，あるハードウェア上で特定の 2 種類のセキュリティ機能を組み合わせてはいけない, というものがあり，それに該当する場合などである．このようなセキュリティ機能の組合せは, 表層的矛盾検出では検出することができない.

意味的な矛盾や不具合を判定するために, タプルの組み合わせとそのタプルの中身であるセキュリティ機能を指定して意味的に矛盾であるとする条件を表現した制約知識を用意しておく.

意味的矛盾として，Prohibition（組合せ禁止）と Lack（組合せ欠如）を導入する.

Prohibition は，ある特定の 1 つ以上のセキュリティ機能が同時に実現してはいけないに

も関わらず実現している状態である．例えば，パケットフィルタリングの機能があるハードウェア上の 2 つ以上のソフトウェアで実現されている場合などである．単純なパケットフィルタリング機能は1つのハードウェア上には1つあれば十分であるためである．

Lack は，同時に実現することが望まれるセキュリティ機能が同時に実現されていない状態である．例えば，ログを取得する場合は時刻合わせをしておくことが推奨されるが，ログを取得するルールはあるが，時刻合わせを行うルールがない場合などである．

図 8 に制約知識の例を示す．制約知識は，対象とするセキュリティ機能の組合せと，それらの機能が実現される場所（ハードウェア上かセグメント上）で条件を表現している．また，constraint タグの type 属性で意味的矛盾の種類を指定している．

```
<constraints>
    <constraint name="c001" type="prohibition" where="network.segment.*"
     region="in">
        <comment>ログ取得は改竄防止に影響する</comment>
        <function action="add">function.integrity</function>
        <function action="add">function.logging</function>
    </constraint>
    <constraint name="c002" type="lack" where="network.*" region="over">
        <comment>ログ取得は全てでしなければならない</comment>
        <function action="add">function.logging</function>
    </constraint>
    <constraint name="c003" type="lack" where="network.*" region="in">
        <comment>ログ取得する場合は時刻を合わせておく</comment>
        <function action="add" sufficient="true">function.logging</function>
        <function action="add">function.synchronize.time</function>
    </constraint>
</constraints>
```

図 8．制約知識の例

機能マッピング結果の全タプルを走査して意味的矛盾を検出する．Prohibition の場合は，制約知識の条件に合致した場合が矛盾であり，Lack の場合は，制約知識の条件に合致しなかった場合が矛盾となる．

4.5 意味的矛盾の解消

前節の矛盾判定式によって，意味的矛盾が検出できる．これらの矛盾を解消する方法について示す．

Prohibition は，推奨されないタプルの組み合わせであるため，この矛盾を解消するには，ノードを削除するか，ルールを削除する．ただしこの場合，これらの削除によって表層的矛盾が発生する場合がある．

Lack は，存在すべきタプルの組み合わせであるため，この矛盾を解消するには，ルールを必要に応じて追加した上で，対応するノードを追加する．

4.6 意味的矛盾検出と解消の動作例

意味的矛盾検出に関しても実装を行った．図 9 は，意味的矛盾検出を行った画面ダンプの例であり，意味的矛盾検出によって，Lack が存在することを示している．

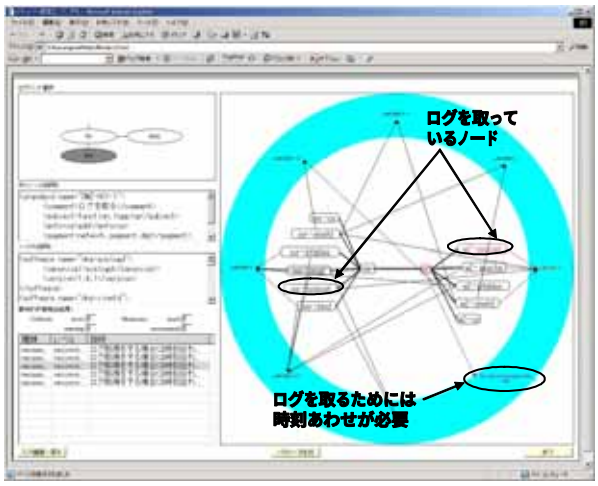

図 9．意味的矛盾の検出

5 おわりに

策定されたセキュリティポリシーと対象ネットワークシステムのシステム構成を対応付けることでセキュリティポリシーの矛盾を検出する機能マッピング方式と矛盾検出方式について述べた．表層的な矛盾だけではなく，意味的な矛盾も検出できる枠組みである．また，試作したシステムで実際に矛盾を検出し，修正を支援することができることを確認した．

参考文献

[1] BS7799: Code of practice for information security management, 1999.
[2] 財団法人日本情報処理開発協会，「情報セキュリティマネジメントシステム(ISMS)適合性評価制度」, http://www.isms.jipdec.jp/,.
[3] 松田他，「機能マッピングによるセキュリティ対策状況の把握」，情報処理学会研究報告 Vol.2004，No.129，pp.95-100，2004.
[4] 岡城他，「セキュリティ運用管理のためのセキュリティポリシー言語 SCCML」，情報処理学会研究報告 Vol.2004，No.129，pp.89-94，2004.