2005－DSM－36（13）
2005／3／18

ステートチャートによる仕様ベース侵入検知モデルの提案

渡辺 裕太
千葉大学工学部情報画像工学科

今泉 貴史
千葉大学総合メディア基盤センター

侵入を早期に発見し被害を最小限に食い止めるために様々な侵入検知手法が提案されている。仕様ベース侵入検知は、監視対象の望ましい振る舞いをあらかじめ記述しておき、監視対象が記述から逸脱した振る舞いをしたときに侵入と判定する手法である。本論文では、新しい仕様ベース侵入検知モデルである DS-Model を提案する。DS-Model では、Unified Modeling Language(UML) のダイアグラムの一つであるステートチャート図を用いて望ましい振る舞いを記述する。DS-Model は、未知、既知問わずに侵入検知が可能であり、false negative や false positive が少ない。また、既存の仕様ベース侵入検知と比較し記述が理解しやすい。

A Proposal of Specification-based Intrusion Detection Model with Statechart

Yuta WATANABE
Department of Information & Image Science,
Chiba University

Takashi IMAIZUMI
Institute of Media and Information Technology,
Chiba University

Many approaches have been proposed to detect intrusions. One of them is specification-based intrusion detection. It describes desirable behavior of programs. Deviations from desirable behavior are flagged as intrusions. In this paper we propose new specification-based intrusion detection model, DS-Model. We describe desirable behavior with statechart diagram of Unified Modeling Language. DS-Model can detect unknown and known attacks. Rates of false negatives and false positives are low. It is easyer than other specification-based intrusion detection to understand description.

1　はじめに

現在では多くの計算機がネットワークに接続されている。計算機をネットワークへ接続すると、ウィルスやクラッカーに侵入される可能性がある。本論文では、計算機への “侵入” を、プロセスに運用者の想定と異なる振る舞いをさせることととらえる。早期に侵入を発見し被害を最小限に抑えるために、侵入検知システムが普及してきている。

侵入検知を行うための手法は様々なものが提案されている。攻撃をシグネチャとして記述する手法は、仕組みが単純で負荷が小さいことや、利用者にとって理解しやすいことから多くの侵入検知システムで採用されている。しかし、この手法は未知の攻撃を検知できない。また、攻撃のパターンを少し変化させるだけで検知不可能になる場合がある。

統計的手法や学習によって正常時と侵入時の違いを検知する手法は未知の攻撃を検知できる [1][2]。しかし、学習期間中に正常時の振る舞いを全て学習することが難しく、false positive[1]が生じやすい。

ソースコードを静的解析し、プロセスがソースコードによって規定されている制御フローに従っているかを監視する手法は、未知の侵入を検知でき、かつ false positive が生じにくい [3]。しかし、ソースコードによって規定される制御フローを逸脱することなく侵入された場合は、侵入を検知することができない。

本論文では次の要件を満たす侵入検知モデルを提案する。

(1) 未知の攻撃を検知できる

(2) false negative[2]と false positive が少ない

(3) 記述が理解しやすい

(1) によって、近年増加している zero-day attack[3]を検知できる。(2) によって、侵入を見逃す可能性を減らすことと、管理者の負担を減らすことができる。(3) によって、侵入検知システムの専門家がいない企業や大学でも利用が可能になる。

以下、2 章では仕様ベース侵入検知の概要と、既存の研究について述べる。3 章では本論文で提案する DS-Model を説明する。4 章では DS-Model のプロトタイプ実装である OPERA-System について述べる。5 章では OPERA-System を用いた実験とその評価を述べ

[1]正常な振る舞いを侵入と判定する誤検知

[2]侵入を表す振る舞いを見逃す誤検知

[3]発見されて間もない脆弱性を悪用する攻撃

る。6 章では既存の研究との比較を行う。7 章では本論文をまとめ今後の課題を述べる。

2 仕様ベース侵入検知

仕様ベース侵入検知は、監視対象の望ましい振る舞いをあらかじめ記述しておき、監視対象が記述した内容から逸脱した振る舞いをしたときに侵入とみなす手法である。仕様ベース侵入検知は、いくつかの手法が提案されている。

R. Sekar らは、正規表現を拡張した Regular Expressions over Events (REE) を用いて、システムコールのパターンを記述する手法を提案している [4]。パターンにマッチしたときにアクションを起こすことで、侵入を検知することや侵入を防ぐことができる。しかし、パターンにマッチしないと検知できないので、未知の侵入方法を見過ごしてしまう可能性がある。

C. Ko らは、文脈自由文法を拡張した Parallel Environment grammars(PE-grammars) を用いて望ましい振る舞いを記述し、システムコール列をパーズすることで侵入検知を行う手法を提案している [5]。記述の表現能力は高いものの、侵入検知システムの利用者が必ずしも理解できるとは限らない。

3 DS-Model

3.1 DS-Model の概要

本論文で提案する DS-Model(Detection with Statechart Model) は、侵入を定義するための概念と記述方法を工夫することで既存の仕様ベース侵入検知の欠点を補うモデルである。侵入を定義するための概念として、非決定性有限状態オートマトン (NFSA) を拡張した非決定性階層拡張有限状態オートマトン (NHE-FSA) を用いる。記述には Unified Modeling Language(UML)[6] のステートチャート図を用いる。

DS-Model の概要を図 1 に示す。DS-Model は記述時と侵入検知時に分けられる。記述時は、監視対象とするプログラムを決め、そのプログラムを実行するプロセスの望ましい振る舞いをステートチャート図で記述する。このとき、仕様書やソースコードを参考にする。侵入検知時は、監視対象のプログラムを実行するプロセスが現れた時に NHE-FSA を生成する。そして、そのプロセスが監視対象のプログラムを実行している間に呼び出すシステムコール列を NHE-FSA へ入力する。NHE-FSA がシステムコール列を受理した場合、正常と判定する。NHE-FSA がシステムコール列を拒否した場合、侵入と判定する。

監視中のプロセスが子プロセスを生成した場合、親プロセスの NHE-FSA を複製し、一方を子プロセスに割り当てる。子プロセスの NHE-FSA は親プロセスの NHE-FSA と同じ状態から動作を始める。

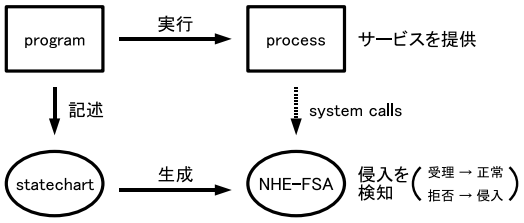

図 1: DS-model の概要

3.2 DS-Model による侵入検知例

cat コマンドを例に DS-Model を説明する。記述時は、ステートチャート図で cat コマンドが呼び出すシステムコール列を受理する NHE-FSA を記述する。図 2 に記述例を示す。ただし、初期化処理に相当する状態 initialization の内部は省略している。

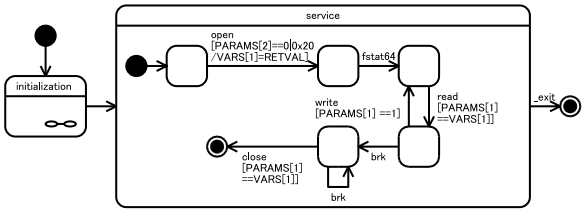

図 2: cat コマンドの望ましい振る舞い

侵入検知時は、cat コマンドを実行しはじめてから終了するまでの間にプロセスが呼び出したシステムコール列を NHE-FSA へ入力する。例えば cat コマンドを実行するプロセスが、初期化処理を終えた後 open, fstat64, read, write, read, brk, close, _exit というシステムコール列を呼び出したとする。この場合、NHE-FSA がシステムコール列を受理するので、正常なプロセスと判定する。cat コマンドを実行するプロセスが、初期化処理を終えた後に open, fstat64, read, execve というシステムコール列を呼び出したとする。この場合、NHE-FSA はシステムコール列の入力途中で動作不可能になり NHE-FSA がシステムコール列を拒否するので、侵入されているプロセスと判定する。

3.3 NHE-FSA の定義

NHE-FSA は、NFSA の状態と記号をそれぞれ入れ子拡張状態と拡張状態へ拡張したものである。

入れ子拡張状態 $\mathbf{q}$ は、状態、n 個の状態変数[4]、子 NHE-FSA 集合[5]の組であり、次のように表せる。

$$\mathbf{q} = \langle q, v_1, v_2, ..., v_n, W \rangle$$

ここで、q と W は一対一もしくは多対一の関係にある。また、再帰的な入れ子は許可しない。

拡張記号 $\mathbf{a}$ は、記号と m_a 個の引数[6]の組であり、次のように表す。

$$\mathbf{a} = \langle a, p_1, p_2, ..., p_{m_a} \rangle$$

非決定性階層拡張有限状態オートマトン M を形式的に次の 5 つ組で定義する。

$$M = \langle \mathbf{Q}, \mathbf{\Sigma}, \delta, \mathbf{q}_0, \mathbf{F} \rangle$$

- $\mathbf{Q}$ は入れ子拡張状態の有限集合
- $\mathbf{\Sigma}$ は入力拡張記号の有限集合
- δ は遷移関数 ($\delta : \mathbf{Q} \times (\mathbf{\Sigma} \cup \epsilon) \rightarrow 2^{\mathbf{Q}}$)
- $\mathbf{q}_0$ は初期入れ子拡張状態
- $\mathbf{F}$ は終了入れ子拡張状態集合

である。

NHE-FSA が拡張記号を入力したときの動作は、次の 2 種類がある。

(1) 遷移関数に基づいて状態遷移する。このとき、遷移先を非決定的に選択する場合がある。

(2) 子 NHE-FSA 集合から一つの NHE-FSA を非決定的に選択し記号を入力する。

どちらを選択するかは現在の入れ子拡張状態に依存する。現在の入れ子拡張状態が子 NHE-FSA を持たない場合は (1) を選択する。現在の入れ子拡張状態が子 NHE-FSA を持ち、かつ、いずれかの子 NHE-FSA が終了状態ではないときは (2) を選択する。現在の入れ子拡張状態が子 NHE-FSA を持ち、かつ、全ての子 NHE-FSA が終了状態のときは (1) か (2) を非決定的に選択する。

ある拡張記号列を NHE-FSA へ入力し終えたとき NHE-FSA が終了状態であれば、NHE-FSA はその拡張記号列を受理する。拡張記号列を入力している途中で NHE-FAS が動作不可能になった場合や、拡張記号列を入力し終えたときに NHE-FSA が終了状態でない場合は、NHE-FSA はその拡張記号列を拒否する。

[4]数値や文字列を格納する変数
[5]NHE-FSA の有限集合
[6]数値や文字列、構造体を格納する変数

3.4 NHE-FSA の記述

NHE-FSA の記述は、表 1 に示す NHE-FSA の要素とステートチャート図の要素の対応関係に基づき記述する。NHE-FSA と対応関係のない要素は記述しない。

表 1: 対応表

NHE-FSA の要素	UML の要素
状態 (q)	状態もしくは合成状態
子 NHE-FSA($M' \in W$)	並行領域内の状態機械
状態変数 (v_i)	状態機械所有オブジェクト
記号 (a)	イベント
引数 (p_i)	引数
動作関数 (δ))	遷移
初期状態 (q_0)	初期擬似状態から遷移した先の状態
終了状態 (F)	最終状態

ここで表 1 を補足する。状態変数や引数は明記しない。遷移関数を記述するときは、ガード条件で引数と状態変数に対する条件を記述し、動作式で状態変数に対する代入式を記述する。このとき、引数は `PARAMS[i]`、状態変数は `VARS[i]` と表記する。ガード条件や動作式では表 2 の定数を用いることができる。また、一般的な演算子に加え、パターンマッチングを行う演算子`=~`や、ファイルがディレクトリ内に存在するか評価する演算子`=^`を用いることができる。終了状態は、最終状態へイベントの無い遷移を持つ状態で表してもよい。

表 2: 定数

表記	意味
PID	プロセス ID
UID	ユーザ ID
GID	グループ ID
EUID	実効ユーザ ID
EGID	実効グループ ID
CWD	カレントディレクトリ

4 OPERA-System

DS-Model を Linux 上で実装し OPERA-System(Observing Process Events to be Rejected and Accepted System) を開発した。OPERA-System はオープンソースソフトウェアである syscalltrack[7] に機能を追加する形で実装した。コメントを含め C 言語と C++によるプログラムを約 5000 行追加した。

4.1 構成

OPERA-System の構成は syscalltrack の構成をほぼ踏襲している。構成を図 3 に示す。

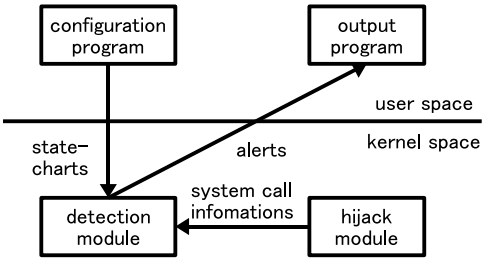

図 3: OPERA-System の構成

設定プログラムは、ステートチャート図を読み込み、検知モジュールへ入力するプログラムである。実装を容易にするためステートチャート記述言語を定義し、その記述を設定プログラムが読み込む形で実装を行った。

ハイジャックモジュールは、任意のプロセスによってシステムコールが呼び出されたとき、本来のシステムコールを実行する前後で検知モジュールにシステムコールの情報を通知するモジュールである。

検知モジュールは、OPERA-System の中核的なモジュールであり、NHE-FSA のシミュレートを行い侵入検知を行う。侵入を検知した場合、警告を出力する。詳細は後で述べる。

出力プログラムは、検知モジュールが出力した警告を管理者に通知するプログラムである。

4.2 検知モジュール

検知モジュールは、ハイジャックモジュールからシステムコールの情報を通知され、その情報に基づき後で述べる処理を行う。実装上の都合から execve, _exit システムコールは本来のシステムコールを実行する直前で処理を行い、他のシステムコールは直後に処理を行う。

まず、システムコールを呼び出したプロセスに対応する NHE-FSA が存在しない場合の処理の概要を述べる。

(1) 直前に呼び出したシステムコールが execve システムコールであり、かつ成功した場合は、新しいプログラムに対応する NHE-FSA を生成する。

(2) execve システムコールが呼び出された場合はフラグを立てる。(1) はこのフラグを参照している。

次に、システムコールを呼び出したプロセスに対応する NHE-FSA が存在する場合の処理の概要を述べる。

(1) 直前に呼び出したシステムコールが execve システムコールであり、かつ成功した場合は、NHE-FSA の受理判定を行い結果を出力した上で NHE-FSA を削除し、新しいプログラムに対応する NHE-FSA を生成する。

(2) NHE-FSA に 3.3 で述べた動作をさせる。非決定的な選択が必要な場合は、現在の入れ子拡張状態を複製し、それぞれ別の選択をする。NHE-FSA が動作不可能である場合は警告を出力する。

(3) execve システムコールが呼び出された場合はフラグを立てる。(1) はこのフラグを参照している。

(4) _exit システムコールが呼び出された場合は、受理判定を行い結果を出力した上で、NHE-FSA を削除する[7]。

(5) fork, vfork システムコールが呼び出された場合は、親プロセスの NHE-FSA を複製し一方を子プロセスに割り当てる。

5 実験と評価

DS-Model が侵入検知に有効であることを確かめるために実験を行った。実験は下記の 2 つのプログラムを監視対象とし、正常時と侵入時の OPERA-System のログを確認する形で行った。

(1) バッファオーバフローの脆弱性を持つエコーサーバ

(2) 入力チェックに脆弱性を持つ簡易ファイル転送サーバ

(1) はクライアントから受信したデータをそのまま送り返すプログラムである。クライアントから大きなデータを受信すると、バッファにバッファサイズ以上のデータを書き込んでしまう。この脆弱性を利用すると、侵入者は任意のコードを実行できる。バッファオーバフローの脆弱性は悪用されることが多いため、このプログラムを用いた実験を行った。

(2) はクライアントの要求に応じてホームディレクトリ内のファイルをクライアントへ転送するプログラムである。ホームディレクトリ外のファイルを送信しないようにするために要求されたパスの入力チェックを行う。しかし、入力チェックの実装が甘く、侵入者は親ディレクトリを表す文字列 .. を用いることでホームディレクトリ外のファイルを転送させることが可能

[7] プロセスは、_exit システムコールを呼び出さなくても終了することがある。例えば、KILL シグナルを受信した場合である。このような場合、現在の実装では直接的にプロセスの終了を捕らえることができない。そのため、プロセス ID が再利用された時点でプロセスが終了したとみなし、_exit システムコールが呼び出されたときと同様の処理を行っている。

である。プログラムの実装ミスを侵入者に悪用されることを想定し、このプログラムを用いた実験を行った。

実験環境として、サーバ[8]とクライアント[9]の 2 台の計算機を用意しスイッチングハブを介して接続した。サーバへは (1) と (2) のプログラムと OPERA-System をインストールした。また、(1) と (2) それぞれの望ましい振る舞いを記述し OPERA-System へ入力した。クライアントへは (1) と (2) それぞれに対応した正常時テストプログラムと侵入プログラムをインストールした。

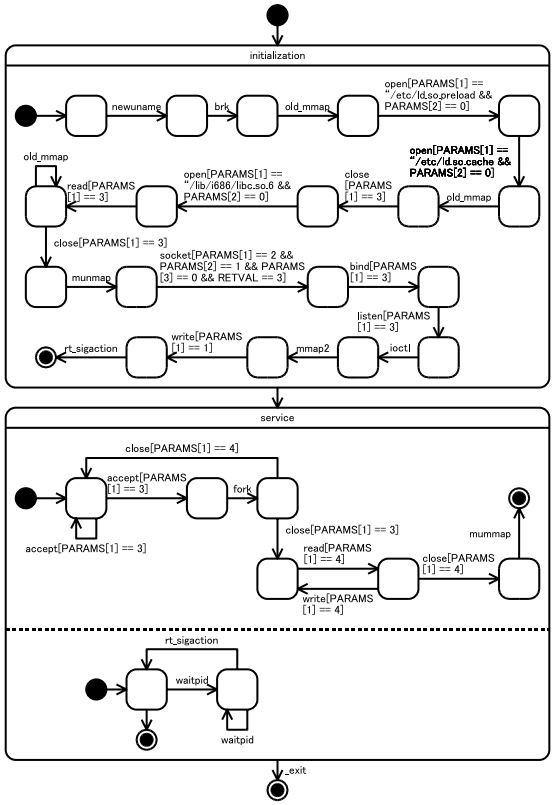

図 4: エコーサーバの望ましい振る舞い

(1) を監視対象とした実験について述べる。この実験では、エコーサーバの望ましい振る舞いとして図 4 を用いた。まず、正常時テストプログラムを 5 万回実行し、エコーサーバとデータを送受信することを繰り返した。このとき、エコーサーバの子プロセス全てが正常なプロセスであると判定された。次に、侵入プログラムを実行し、バッファオーバフローの脆弱性を利用しシェルを起動させた。このとき、ローカル変数の値が書き換えられたことにより、直後に呼び出したシステムコールの引数の値が正常時と異なったため、侵入と判定された。以下に OPERA-System のログを示す。

```
551["echod"]: 4_write(-1610614428, "\144\144\144
551["echod"]: 3_read(-1610614428, "\144\144\144
551["echod"]: 6_close(-1610614428) = -9
551["echod"]: 63_dup2(4, 0) = 0
551["echod"]: 63_dup2(4, 2) = 2
551["echod"]: 63_dup2(4, 1) = 1
551["echod"]: 11_execve("/bin/sh", 9ffffa96,
551: invalid.
```

(2) を監視対象とした実験について述べる。まず、正常時テストプログラムを 5 万回実行し、ホームディレクトリ内にあるファイルを繰り返し転送させた。このとき、簡易ファイル転送サーバの子プロセス全てが正常なプロセスであると判定された。次に、侵入プログラムを実行し、パスワードファイルを転送させた。このとき、簡易ファイル転送サーバのプロセスが、ホームディレクトリ外のファイルをオープンした時点で侵入されていると判定された。以下に OPERA-System のログを示す。

```
857["catd"]: 5_open("../../etc/passwd", 0, 438)
857["catd"]: 90_old_mmap(mmap_arg_struct{0, 4096
857["catd"]: 3_read(3, "root:x:0:0:root:/root:
857["catd"]: 4_write(4, "root:x:0:0:root:/root:
857["catd"]: 4_write(4, "bin:x:1:1:bin:/bin:/
(以降省略)
```

実験の評価について述べる。いずれのプログラムを監視対象とした場合も false positive は発生しなかった。統計的手法や学習を用いずに、プログラムの仕様やソースコードを参考に望ましい振る舞いを記述するためと考えられる。

また、実験では望ましい振る舞いと侵入時の振る舞いの違いから侵入を検知することができた。このことから、未知の攻撃を検知可能であるといえる。

エコーサーバを監視対象とした実験では、侵入者が送り込んだコードを実行させる侵入を検知することができた。また、簡易ファイル転送サーバを監視対象とした実験では、悪意のあるコードを実行させることなく開発者の想定と異なる振る舞いをさせる侵入を検知することができた。様々な侵入を検知可能であり、false negative が少ないといえる。

6 関連研究との比較

本章では既存の仕様ベース侵入検知との比較を行う。

REE による手法は、パターンにマッチしなければ正常と判定するため、未知の攻撃を見過ごしてしまう可能性がある。一方、DS-Model は、パターンにマッチしなければ侵入と判定する。記述者が望ましいと考え

[8]CPU は Celeron 300MHz、メモリは 64Mbites、ネットワークは 100Base-TX、OS は Vine Linux 3.1

[9]CPU は Duron 1000MHz、メモリは 384Mbites、ネットワークは 100Base-TX、OS は Vine Linux 3.1

る振る舞い以外は侵入と判定するため、未知の攻撃を見逃してしまうことはない。そのため、REE による手法に比べて DS-Model は false negative が少ないと考えられる。

PE-grammars は文脈自由文法を拡張したものであり、文脈依存文法の能力をもっている。そのため、PE-grammars は NHE-FSA より表現力の点で勝っている。ここで、PE-grammars による記述例を文献 [8] から引用する。

```
SPEC rdist (<?, rdist, *, *>)
    ENV User U = getuser();
    ENV int NODEID = 0;
    SE: <rdist>
    <rdist> -> <command> <rdist> | .
    <command> -> <up_file> | <up_slink> | <up_dir>.
    <up_file> -> creat { NODEID = F.nodeid; }
                 <opt_chmod>
                 <opt_chown>
                 <efile>.
    <efile> -> rename-NODEID | unlink-NODEID.
    <up_slink> -> symlink { NODEID = F.nodeid; }
                  <efile>.
    <makedir> -> (mkdir) { NODEID = F.nodeid; }
                 <opt_chmod>
                 <opt_chown>.
    <opt_chmod> -> chmod-NODEID.
    <opt_chown> -> chown-NODEID-U | .
END;
```

PE-grammars による記述は、構文解析に関する知識が無い者にとっては理解しにくい。DS-Model では同等な内容を図 5 のように表すことができる。DS-Model は図式記法を採用しているので PE-grammars と比較して処理の流れを視覚的に捕らえやすい。また、UML はシステム開発において広く用いられているので認知度が高い。以上のことから、より多くの人が理解できると期待できる。

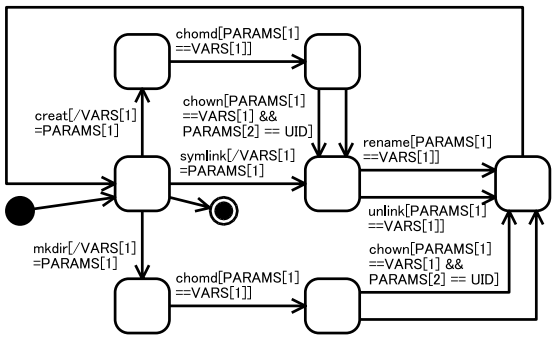

図 5: rdist の望ましい振る舞い

7 おわりに

本論文では、新しい仕様ベース侵入検知である DS-Model を提案した。DS-Model は侵入を定義するための概念として非決定性階層拡張有限オートマトンを用い、記述にはステートチャート図を用いる。正常時の振る舞いと侵入時の振る舞いの違いから侵入を検知するため、未知、既知問わずに侵入を検知できる。プログラムの仕様やソースコードを参考に望ましい振る舞いを記述するため false positive が少ない。人手で望ましい振る舞いを記述するので、ソースコードの制御フローを逸脱することなく侵入された場合でも侵入を検知できる。また、広く利用されている図を用いるため、記述は多くの利用者が理解できると期待できる。

今後の課題は、ステートチャート図の半自動生成である。これによって利用者の負担を低減することができる。また、利用者が生成されたステートチャート図を確認、修整する過程でプログラム中のバグを発見できる可能性がある。

長期的な課題は、SE-Linux の登場により広く知られるようになった強制アクセス制御との統合である。強制アクセス制御は、多くの場合システムコールの前後関係は考慮しない。しかし、DS-Model はシステムコールの前後関係を考慮するので、既存の強制アクセス制御と比較しより厳密なアクセス制御ができる可能性がある。

参考文献

[1] Sato, I., Okazaki, Y. and Goto, S.: An Improved Intrusion Detecting Method Based on Process Profiling, *IPSJ Journal*, Vol. 43, No. 11, pp. 3316–3326 (2002).

[2] Sekar, R., Bendre, M., Dhurjati, D. and Bollineni, P.: A Fast Automaton-Based Method for Detecting Anomalous Program Behaviors, *SP '01: Proceedings of the IEEE Symposium on Security and Privacy*, IEEE Computer Society, p. 144 (2001).

[3] 阿部洋丈, 大山恵弘, 岡瑞起, 加藤和彦: 静的解析に基づく侵入検知システムの最適化, 情報処理学会論文誌, Vol. 45, No. SIG03.

[4] Uppuluri, P. and Sekar, R.: Experiences with Specification-Based Intrusion Detection, *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, Springer-Verlag, pp. 172–189 (2001).

[5] Ko, C., Ruschitzka, M. and Levitt, K.: Execution monitoring of security-critical programs in distributed systems: a Specification-based approach, *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, IEEE Computer Society, p. 175 (1997).

[6] Group, O. M.: UML 仕様書, 株式会社アスキー (2001). OMG Japan SIG 翻訳委員会 UML 作業部会.

[7] : syscalltrack. http://syscalltrack.sourceforge.net/.

[8] Ko, C.: Execution monitoring of security-critical programs in distributed systems: a Specification-based approach, *PhD thesis, Department of Computer Science, University of California, Davice* (1996).