プロセス実行履歴に基づくアクセスポリシー自動生成システム

原田 季栄　保理江 高志　田中 一男
(株)NTTデータ 技術開発本部
e-mail: {haradats, horietk, tanakakza}@nttdata.co.jp

概 要

強制アクセス制御機構 (Mandatory Access Control) の導入は、コンピュータシステムのセキュリティを保つ上で有効な手法のひとつであるが、アクセス定義（制御）の粒度に比例した量のポリシーを「過不足なく」設定し、管理運用することがその前提となる。本論文では、カーネルにプロセスの実行履歴を記録する機構を組み込むという手法により半自動的に必要かつ適切なポリシーを設定するための手法について解説し、筆者らがLinuxにおいて構築したプロトタイプを紹介する。

Access policy generation system based on process execution history

Toshiharu HARADA　Takashi HORIE　Kazuo TANAKA
Research and Development Headquarters, NTT DATA CORPORATION
e-mail: {haradats, horietk, tanakakza}@nttdata.co.jp

Abstract

MAC (Mandatory Access Control) has the ability to improve security of Linux operating system dramatically. However, defining and managing proper policy is not easily achieved because program dependencies are usually invisible from system administrators. This paper presents the challenges in providing automatic policy generation based on process execution history.

1. はじめに

コンピュータシステムのセキュリティを考えるとき、適切なアクセス制御の実現は中核としての役割を持ち、早くからオペレーティングシステムにおけるセキュリティモデルと実装の研究が行われている。アクセス制御機構の指標としては、アクセス制御の単位の細かさ(granularity)とアクセス制御の強制的執行(enforcement)があげられ、それらを Linux OS 上で実現した NSA (National Security Agency、米国家安全保証局)の SELinux (Security-Enhanced Linux)[1, 2]が注目を集めている。

筆者らは SELinux や SubDomain[3], RSBAC[4]等の Linux への強制アクセス制御の実装の評価を行う過程で、細粒度の高いアクセス制御を Linux に適用する場合、オペレーティングシステムの基本構造に由来する本質的な困難さが存在し、それが最終的には「必要十分なポリシーの定義」という作業で問題として具体化することに気がついた。

本論文では、Linux で強制アクセス制御を実現する際の運用上の課題とその背景について解説し、その課題を解消するための一つの手法として筆者らが構築した SELinux 向けアクセス制御ポリシーの自動生成システムについて紹介する。

2. 強制アクセス制御

2.1. Linuxにおけるアクセス制御の実装

Linuxに標準的に実装されているアクセス制御は、ファイルやディレクトリの所有者が、所有者自身/グループ/それ以外の3つのカテゴリに分類されたユーザに対するアクセス許可を「読みこみ、書きこみ、実行」（ディレクトリに対しては「読みこみ、書きこみ、検索」）のパターンの組み合わせとして指定し、その結果を対象のアクセス許可情報として管理するというものである。アクセス要求の認可は、Linuxの母体であるUNIXの伝統を引き継ぎ、当該アクセス要求を行ったプロセスがシステム管理者権限で動作している場合には無条件に認可するようプログラムされているため、標準の

Linuxではクラッカーにより不正にシステム管理者権限を奪われると致命的な被害を免れることができず、Linuxにおける脆弱性として知られている。

2.2. 強制アクセス制御の概要とLinuxへの適用効果

強制アクセス制御 (Mandatory Access Control) は、1985 年に DoD (Department of Defense, 米国防総省)が発行した TCSEC (Trusted Computer System Evaluation Criteria) [5]の B1 クラス、”LABELED SECURITY PROTECTION”の中核となる機能としてとりあげられている。

強制アクセス制御は、個々のアクセス要求毎にあらかじめ定義されたポリシー情報による認可が与えられた上ではじめて実行される点が特徴であり、それ自体は特定の OS に依存しない汎用的なアプローチである。この手法は前述のように単純すぎるアクセス制御モデルとそれに起因する脆弱性を持つ Linux に対して適用した場合その効果が顕著であり、商用高信頼 OS や OS のセキュリティ強化の試みでは例外なく含まれている。

2.3. 強制アクセス制御の課題

OSへの強制アクセス制御の実装は大きく次の3つの部分に分けられる。

(a) カーネルプロセス内でのアクセス要求に対する条件分岐の追加

(b) ポリシーの保持機構

(c) ポリシーに基づく個々のアクセス要求の認可機構

強制アクセス制御の導入はその位置づけ上OSプログラムへの大規模な改造を伴い、システム全体の処理パフォーマンスへの影響は避けられない。また、導入時の必要十分なポリシーの定義という作業の負担が重要な課題として表面化する。それらが強制アクセス制御に関して一般的に認識されている課題（これらは強制アクセス制御の性質に基づくものであるから問題と称するのは正しくない）である。当然ではあるが、個々のアクセス要求が正しいかどうかは自動的に判断できるものではない。強制アクセス制御はポリシーに基づく強制的なアクセス制御という「機能」あるいは「手段」を提供するが、それを正しく動作させるためには必要なアクセス要求に対する認可を正しく定義しなければならず、不必要なアクセスを認めればセキュリティのレベルは下がるし、逆に必要なアクセスを排除してしまうとサービス自体が動作しないこともあり得る。

2.4. Linuxへの適用

Linuxでは、アプリケーションプロセスからのシステムに対する要求は最終的にはシステムコールとしてカーネルプロセスの中で処理されるため、通常はシステムコールが強制アクセス制御の執行場所として選ばれる。すなわち、システムコール毎にそれが呼び出された時点でアクセス認可を行うことにより、もれなくアプリケーションプロセスの挙動を監視、制御することができる。Linux標準のファイルやディレクトリのアクセス情報は「セキュリティビット」としてファイルシステムの属性として管理されているが、強制アクセス制御では認可するアクセス要求の範囲はファイルだけでなく、またアクセス要求判定の条件も複雑になるためアクセス制御の定義情報はそれを新規にカーネルが管理するデータ構造に追加しカーネルプロセスから参照できるように実装される。

筆者らはLinuxへの強制アクセス制御の導入の最大の問題点は、ポリシーの管理運用、特に「必要十分なポリシーの定義」にあるのではないかと考えている。Linuxにおける基本的な動作スタイルは、C言語により特定の処理を実装し、それをシェルスクリプト等によりOSが標準的に提供しているコマンド群と「組み合わせる」ことにより機能やサービスを提供するというものである[6]。通常、システム管理者は自分が実行しているサービスやプログラムが「最終的にどのような部分により構成され実現されているか」を知る必要はないが、強制アクセス制御を導入し、不適切なアクセス要求を拒否するためには、サービスやプログラムを実現しているコマンド群を構成するファイル毎にそれがどのリソースにアクセスして良いかを掌握し、必要十分なアクセス要求の許可リストをポリシーとして記述しなければならない。

ポリシー定義の課題についてこれ以上詳細に立ち入るのは避けるが、「ポリシーの評価基準」は一意に存在しないことも指摘しておきたい。仮にカーネルの許可要素と動作させるアプリケーションの構成と必要なアクセス許可を把握していたとしても、作成するポリシーの内容は実現したい保護のレベルにより単一の解は存在しない。

強制アクセス制御を導入したLinuxシステムを運用することの困難さは、定義したポリシーが必要十分なアクセス許可を含んでいるかどうかを判断しがたい点にある。

3. SELinux

筆者らが考案、開発したLinuxにおけるポリシー定義の改善について説明するために強制アクセス制御の実装例としてSELinuxをとりあげる。SELinuxで導入された強制アクセス制御は、NSAが研究開発していたFlask[7]に基づいており、アクセスの主体であるプロセス(群)およびアクセスの対象となる資源にラベルを割り当て、ラベルに基づきアクセスを規定する。SELinuxでは、アクセス主体であるプロセスにつけるラベルを「ドメイン」と呼ぶ。

Linux上で提供するweb, ftp等実際のサービスは複数のプログラムの連携として実現されることが多いため、SELinuxではドメインの遷移(プロセス間の起動関係)を定義し、それに基づきアクセス制御を行うよう実装されている。SELinuxの概要については紙面の都合によりIPA等の情報[8, 9]を、ポリシー定義の詳細については開発者による論文[10]をそれぞれ参考とされたい。

3.1.1. SELinuxにおける強制アクセス制御の特徴

SELinuxにおけるポリシー定義は大きく以下のように分類される。

(a) ドメインの定義とそのドメインに対して許可するアクセス対象範囲の定義

(b) ドメインとアクセス許可対象をラベルにより結合

(c) ドメイン間の遷移の定義

SELinuxにおけるポリシー定義も前述の強制アクセス制御の課題の例外ではない。SELinuxにおけるポリシーの定義には、以下の困難さが伴う。

- 細分化されたパーミッションを扱いやすい単位にグループ化する際の基準、およびドメインとして定義するプログラムの対象と範囲を管理者が自分の裁量で決定しなければならない。
- 利用するサービスに必要なアクセス許可を洩れなく調査しなければならない。

  SELinuxでは作業を支援するためのユーティリティとして、カーネルでのエラーメッセージから対応するアクセス許可を生成するスクリプトが用意されており、当該アクセス要求を認可するための設定を得ることができるが、最終段階のチューニングには有効でも導入時点の設定が妥当であるかどうかの判断には使えない(アクセス許可が不足している分にはエラーを見て追加できるが、過度にアクセス許可を与えているかは知りようがないため)。
- ドメイン単位でしかアクセス許可を設定できず、ドメインにはそれまでのドメインの遷移履歴情報が含まれていないため、呼び出し履歴に基づいてアクセス許可範囲を限定することができない。そのため、複数の呼び出し元から遷移するドメインに対しては、アクセス許可範囲を和集合として定義する必要がある。

  SELinuxでデフォルトとして配布されているポリシーファイルでは、シェル(Linuxにおけるコマンドインタプリター)についてshell_exec_tというドメインが定義されており、通常はそのドメインへの遷移を必要最小限に抑えるという形でアクセス制御を記述する。シェルはLinuxのコマンドの中でも特によく使われるプログラムであり、セキュリティホールなどを攻撃されて不用意にシェルが実行されると許可された範囲内の全てのプログラムを実行されてしまい非常に危険である。しかし、シェルを禁止すればたちどころにサービスは動作しなくなる。シェルの実行を許可する場合に必要最低限のアクセス許可範囲を与えるためには、「それがどのコンテキストから呼び出されているか」を考慮して、その上でそのコンテキストで最低限必要なアクセス許可範囲を判断しなければならない。例としてシェルをとりあげたが、同じ考え方で全ての必要なプログラムとそれがアクセスするファイルについて考えなければならずそれは著しく困難な作業である。

4. Linuxにおけるポリシー定義の改善

強制アクセス制御の制御対象は実装毎に異なるが、本研究ではほとんどの実装に含まれていて、またポリシー定義の中でも分量的比重が特に高いファイルアクセスに的を絞り、必要十分なポリシー定義を支援するための手法について検討し、評価用のシステムを構築した。アクセス許可の定義(ポリシーの制定)を行う部分までは個別の実装に依存しないが、最終的に出力されるポリシーはSELinuxに基づき説明することとする。

4.1. 目標設定とアプローチ

ポリシーを無駄の少ないものに近づける作業は完全に自動化することは不可能であり、かつ単一無二の解は存在しない。そこで、筆者らはシステムの管理者が必要十分なポリシーを定義するために可能な支援環境を実現すること

を目標とした。支援の方法については、これまで述べてきたLinuxにおける強制アクセス制御ポリシー定義の困難さの要因を解析した結果、実際にプログラムやサービスを走行させながら、発生したアクセス要求をシステムコール単位で抽出、整理し、その結果をシステム管理者に提示しGUIにより確認や修正を施すことにより、もれなく、また確実なポリシーの定義が可能と考えた。

4.2. システム構成

上で述べたアプローチを実現するための構成を図1に示す。

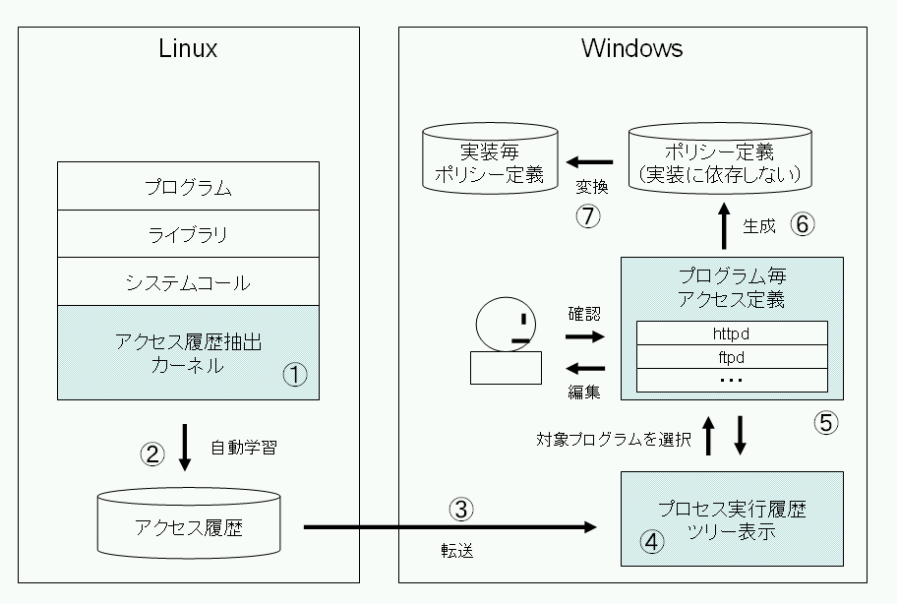

図 1

以下項番に従い動作の流れを説明する。

① ファイルのオープン等アクセス許可を設定可能なシステムコールが呼ばれた際に、呼び出し元のプログラム、対象となるファイルやディレクトリ、アクセスのモード等の情報を記録するよう特殊な修正を施したカーネルを作成し、そのカーネルでwebサービス等のアプリケーションを実行させる(主要な機能について一通り実行する)。

② 前項の結果得られる情報をファイルとして集約する。

③ 得られた情報をポリシー定義を行う端末にftpする。

④ 転送されたアクセス履歴を解析して、/sbin/initに始まるプロセスの起動関係のツリーを構築して、GUIで表示する。

⑤ ツリー情報からアクセス許可を定義したいプログラムを選択し、GUIで当該プログラムのアクセス許可を編集する。アクセス許可の編集は、実際にアクセスされたファイルに対してどのようなアクセス許可を与えるかをリストから選択して行う。アクセス許可編集画面では自動学習された内容に基づき初期値を設定することが可能であり、「実際にプログラムが動作するために必要なアクセス」に近い形から確認、編集を行うことができる。

⑥ 一通り編集が終わったら結果をマージしてポリシー定義ファイルとして保存する。

⑦ 保存されたポリシー定義ファイルを必要に応じて個別の強制アクセス制御実装のポリシー定義の形式に変換する。

項番1および2は本研究が対象としているLinux上でなければならないが、項番3以降については実際にLinux環境である必要性はない。今回GUI開発の効率を考えWindows上で行っている。

5. 実装

5.1. カーネルの修正

5.1.1. アクセス履歴の取得方法と単位

実行履歴の記録の箇所と内容については、SELinux他強制アクセス制御の実装内容を検討した結果表1のように設定し、実際に情報を収集するための仕組みとしてRedHat 8.0をインストールして、ベースのカーネル2.4.20に対して必要な修正を行った。

表 1

ファイル名	場所	記録対象	備考
fs/open.c	sys_open()	rw	ファイルのオープンを検知
	BufferedLog()	アクセスログ	新規作成した関数
fs/exec.c	do_execve()	x	履歴の更新
fs/stat.c	sys_readlink()	l	シンボリックリンクの参照を検知 （履歴の読み出し口としても使用）
kernel/fork.c	do_fork()		履歴の複製
kernel/exit.c	do_exit()		履歴の破棄
include/linux/sched.h	struct task_struct		履歴保持のためのフィールドを追加

履歴の記録を行うためには、SELinuxも対応しているLinuxのセキュリティ拡張フレームワークであるLSM (Linux Security Modules) [11] を使うことも可能であるが、今回はLSMに対応していない強制アクセス制御の実装でも利用可能にするために、標準的なカーネルを修正し用いることとした。ログをとる場合、カーネルプロセスからprintk()で出力されたメッセージをsyslogd 経由で /var/log/messages に保存するというのが一般的であるが、この方法ではsyslogdが動作している間しかメッセージを取得できない上、printk()内部で静的バッファを使用しているため複数のカーネルプロセスが同時に printk()を呼び出した場合にメッセージが破壊されてしまうという問題点があるため、今回はprintk()を使用せず独自に実装している。

BufferedLog() はアクセスログをカーネルメモリ空間内に保存しておくために新規作成した関数である。 内部でkmalloc()を呼び出して128KBの静的メモリ領域を確保し、そこにファイルのオープンやシンボリックリンクの参照が行われた際のアクセスログを蓄えている。蓄えられたアクセスログの読み出しは、sys_readlink()を経由して読み出すことにした。今回はsys_readlink()を読み出し口として使ったが、キーワードを渡すために文字列の引数を持ち、文字列の結果を返すことができるユーザアプリケーションから呼び出し可能なカーネル関数であれば何でも構わない。この方法を用いると、ユーザアプリケーションは既存のカーネル関数を使ってアクセスログを読み出しファイルに保存しているため、修正を加えても「追加・修正された関数名をカーネルの外の世界に公開する必要が無い。」「シャットダウン時にsyslogdが停止した後もファイルシステムが書込み禁止にされるまでのログを残せる。」という利点がある。

Linuxにおいては、全てのプロセスは/sbin/initから派生した子プロセスとして動作する。子プロセスが開始されるときには、現在のプロセスのイメージの複製がsys_fork()によって作成され、複製されたイメージをdo_execve()によって新しいイメージに置き換えるという処理が行われる。

プロセスは task_struct 構造体によって管理されており、sys_fork()の中でこの構造体も複製される。

プロセスが終了するときにはdo_exit()によってこの構造体が破棄される。

プロセス1個に対してこの構造体が1個割り当てられ、プロセスが使用しているメモリやファイルシステムや権限等の情報を保持している。今回はこの構造体に履歴を保持するための変数を文字列（char *）として追加し、sys_fork()でこの履歴を複製、do_execve()で履歴の更新、do_exit()で履歴の破棄を行うように修正を加えた。

sys_open()やsys_readlink()でファイルのアクセス要求を検出した場合、現在の履歴と要求されたファイルの名前およびアクセスモード（読み書き実行）をBufferedLog()に出力するように修正を加えた。出力例を本論文の末尾の付録に示す。

BufferedLog()に蓄えられたアクセスログをログ読み出しのユーザアプリケーションがsys_readlink()経由でほぼリアルタイムに読み出し、ログファイルに保存する。

5.2. フィルタリング

この状態でシステムを運用して得られたログファイルを、アクセスされたファイルが実在するか、実在する場合はファイルかディレクトリかをstat()を用いて確認し、有効なアクセス結果だけをまとめた上でftpによりポリシー編集のためのマシンへ送る。ファイルとディレクトリを区別しているのは、対象により与えられるアクセス許可が異なるためで、ポリシー編集プログラムにおいて対象により表示するメニューを変えるために必要である。

5.3. ポリシー編集用プログラム

ポリシー編集プログラムは「ドメインの追加と削除」「ドメインに対するアクセス許可の設定」「ポリシーの出力」の3つの機能で構成されている。

「ドメインの追加と削除」は、修正したカーネルで自動学習した内容にたまたま特定のプロセスの実行が観測されていなかった場合に追加したり、逆にアプリケーション開発や試験の際に利用したデバッガ等のプログラムについて「サービス提供時のポリシー」から除外するために用いる。「ドメインに対するアクセス許可の設定」は、自動学習されたドメインの一覧から対象となるドメインを選択し、「そのドメインに対してファイルあるいはディレクトリに与えるアクセス許可を編集する」ためのものである。全てのドメインに対して作業を実施することにより必要最小限のポリシーが得られる。最後に「ポリシーの出力」は、確認・編集されたアクセス定義の内容を目的とする強制アクセス制御の処理系のポリシー定義に変換を行う。

5.3.1. プロセス遷移の履歴を用いたドメイン遷移の編集について

図2に、自動学習されたアクセス履歴情報を解析して、プロセスの起動履歴に従ってドメイン遷移をツリー状に表示するプログラムの画面イメージを示す。このツリーで表示されたプロセス遷移の履歴がそのままドメインとなる。

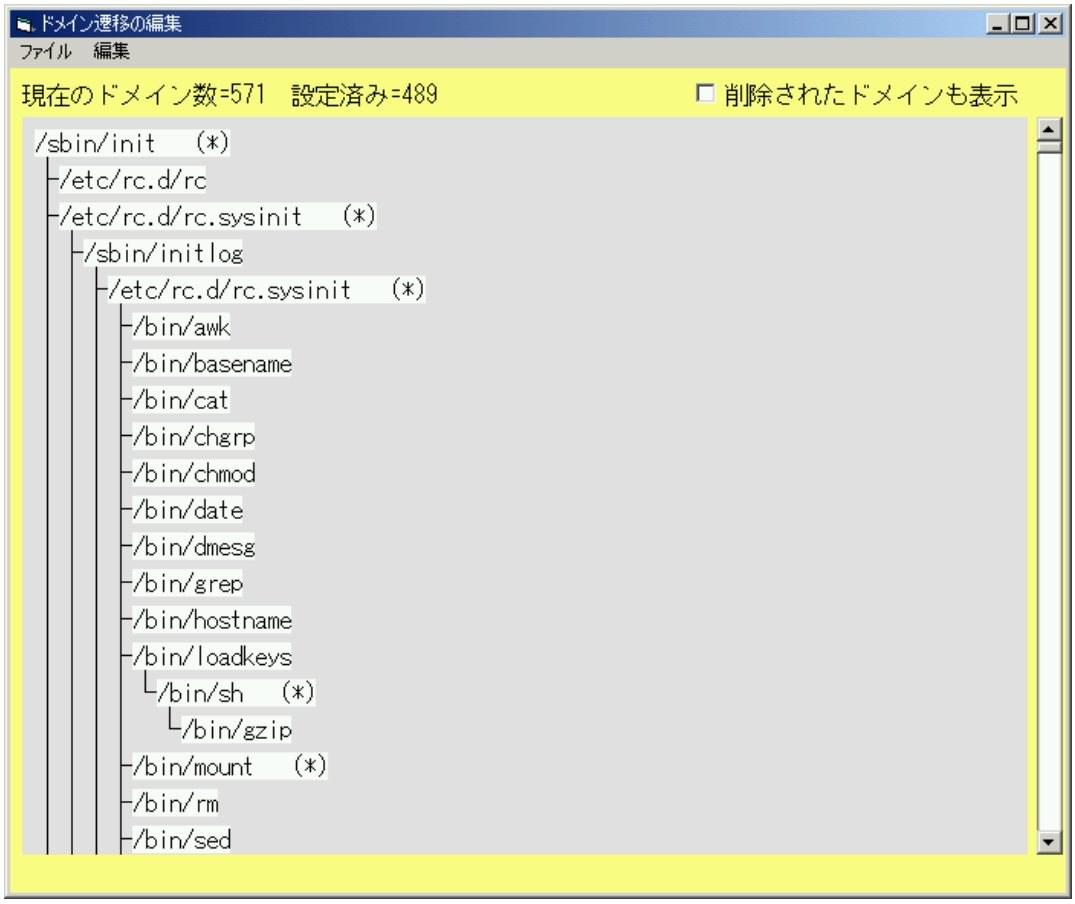

図 2

この画面には、ドメイン遷移の一覧と全ドメイン数とアクセス許可が設定済みのドメイン数が表示される。それぞれのドメインを選択してそのドメインに対するアクセス許可設定（次節を参照）を行う。ドメインの追加や削除は実行可能プロ

グラムの追加と削除に対応する。初期設定の負担を減らすために、条件を指定した自動設定も行うことができる。

ドメインは全て/sbin/initを起点にした履歴により識別され、AというドメインからBというドメインへの一方向の遷移しか存在しない。例えば、/bin/shから/bin/tcshを呼び出し、/bin/tcshから/bin/shを呼び出したとしても、最初の/bin/shと2回目の/bin/shは異なるドメインとして識別されているので、ドメインの遷移が環状になることは無い。

5.3.2. ドメインに対するアクセス許可の設定について

本研究では、Linux OSを対象として汎用の強制アクセス制御の実装用のポリシー定義ファイルを自動的に生成することを目的としている。定義ファイルのベースとするアクセス履歴自動学習の粒度は強制アクセス制御の粒度と同じにすることもできるが、ポリシーの確認、編集を行う利用者に対してはより解りやすい単位での粒度指定が求められる。極端な例として、システムコール毎のアクセス許可を定義させればもっとも細かくアクセス制御を記述できるが、とても現実の運用には耐えないし、そのような実装も存在しない。

今回の開発では、Windows NTFSで指定可能なアクセス権を参考にSELinux向けポリシーにおけるファイルのアクセス権を検討した。表2に指定可能なアクセスパターンを示す。

表 2

ファイル	ディレクトリ
読み込みのみ	読み込みのみ
読み書き	読み書き
実行可能	作成のみ
検出のみ	全て許可
全て許可	全て禁止
全て禁止	

アクセス許可編集プログラムの画面イメージを図3に示す。

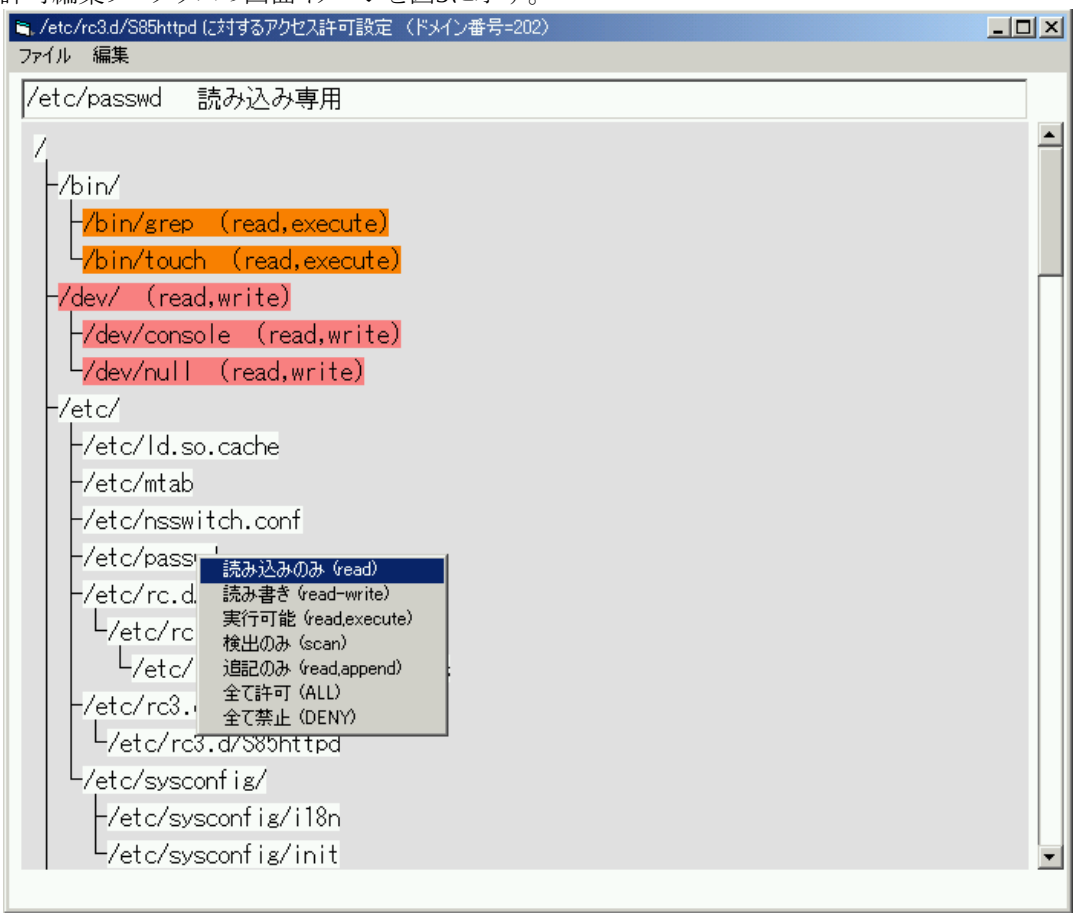

図 3

タイトルバーには編集中のドメインの実行可能プログラムのパスが表示されている。そのドメインに対してアクセスを許可するファイルの一覧がルートディレクトリ(/)を起点としたツリー状に表示される。ファイル名の後ろの括弧内には現在のアクセス許可が表示される。括弧が表示されていないファイルは現在のアクセス許可が「読み込みのみ」に設定されているものである。ファイルやディレクトリに対するアクセス許可を変更したい場合、そのファイルをクリックすると設定可能なアクセス許可の組合せがポップアップ表示されるのでそこから選択する。

5.3.3. ポリシーの出力について

ポリシー出力プログラムに関しては、本論文執筆時点では実際にSELinux用のポリシーを出力するところまで実装できていないが、以下のような方法により変換可能であると考える。出力例も示すが、全てのドメインのアクセス許可設定が終わっていない状態での変換なので、変換されていない部分(file_NOT_FOUND_t)が含まれている。なお、出力例中の#で始まるコメント部分は管理者が理解しやすいように変換プログラムに出力させているものである。

まず、履歴(実行可能ファイル名をコロンで繋いだもの)として管理されているドメイン名を、ポリシーファイルで使用できるドメイン名に変換する。ドメイン名にスラッシュやコロンを含めることはできないので、履歴を連番に対応付ける。

```
type domain_0_t, domain;  # => /sbin/init
type domain_1_t, domain;  # /sbin/init => /etc/rc.d/rc.sysinit => /sbin/initlog => /etc/rc.d/rc.sysinit => /bin/chmod
type domain_2_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /bin/sed
type domain_3_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /bin/sleep
type domain_4_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /etc/sysconfig/network-scripts/ifup-post
type domain_5_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /etc/sysconfig/network-scripts/ifup-post => /bin/basename
type domain_6_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /etc/sysconfig/network-scripts/ifup-post => /bin/grep
type domain_7_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /etc/sysconfig/network-scripts/ifup-post => /bin/hostname
type domain_8_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /etc/sysconfig/network-scripts/ifup-post => /bin/sed
type domain_9_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /etc/sysconfig/network-scripts/ifup-post => /etc/sysconfig/network-scripts/ifup-aliases
type domain_10_t, domain;  # /sbin/init => /etc/rc.d/rc => /etc/rc3.d/S10network => /sbin/initlog => /etc/sysconfig/network-scripts/ifup => /etc/sysconfig/network-scripts/ifup-post => /etc/sysconfig/network-scripts/ifup-aliases => /bin/sed
```

次に、アクセスされるファイルやディレクトリに連番のラベルを付与する。

```
/                          system_u:object_r:file_0_t
/etc/                      system_u:object_r:file_1_t
/etc/ld.so.cache           system_u:object_r:file_2_t
/lib/                      system_u:object_r:file_3_t
/lib/i686/                 system_u:object_r:file_4_t
/lib/i686/libc.so.6        system_u:object_r:file_5_t
```

```
/lib/libacl.so.1              system_u:object_r:file_6_t
/lib/libattr.so.1             system_u:object_r:file_7_t
/lib/i686/libm.so.6           system_u:object_r:file_8_t
/lib/i686/libpthread.so.0     system_u:object_r:file_9_t
/lib/librt.so.1               system_u:object_r:file_10_t
/bin/                         system_u:object_r:file_11_t
/bin/basename                 system_u:object_r:file_12_t
/bin/grep                     system_u:object_r:file_13_t
/bin/hostname                 system_u:object_r:file_14_t
/bin/sed                      system_u:object_r:file_15_t
```

次に、連番に置き換えられたドメインの遷移を定義する。

```
# Allowed process transition from /sbin/init
domain_trans(domain_0_t, file_281_t, domain_111_t)   # /etc/rc.d/rc
domain_trans(domain_0_t, file_620_t, domain_222_t)   # /etc/rc.d/rc.sysinit
domain_trans(domain_0_t, file_NOT_FOUND_t, domain_518_t)   # /sbin/mingetty
domain_trans(domain_0_t, file_NOT_FOUND_t, domain_519_t)   # /sbin/shutdown
domain_trans(domain_0_t, file_NOT_FOUND_t, domain_521_t)   # /sbin/update

# Allowed process transition from /sbin/init:/etc/rc.d/rc
domain_trans(domain_111_t, file_58_t, domain_27_t)   # /etc/rc3.d/S12syslog
domain_trans(domain_111_t, file_77_t, domain_33_t)   # /etc/rc3.d/S13portmap
domain_trans(domain_111_t, file_87_t, domain_39_t)   # /etc/rc3.d/S14nfslock
domain_trans(domain_111_t, file_287_t, domain_47_t)   # /etc/rc3.d/S20random
  （中略）
domain_trans(domain_111_t, file_309_t, domain_350_t)   # /etc/rc6.d/K80random
domain_trans(domain_111_t, file_310_t, domain_358_t)   # /etc/rc6.d/K86nfslock
domain_trans(domain_111_t, file_311_t, domain_366_t)   # /etc/rc6.d/K87portmap
domain_trans(domain_111_t, file_312_t, domain_373_t)   # /etc/rc6.d/K88syslog
domain_trans(domain_111_t, file_313_t, domain_381_t)   # /etc/rc6.d/K90network
domain_trans(domain_111_t, file_62_t, domain_408_t)   # /sbin/consoletype
domain_trans(domain_111_t, file_63_t, domain_409_t)   # /sbin/initlog
domain_trans(domain_111_t, file_280_t, domain_500_t)   # /bin/egrep
domain_trans(domain_111_t, file_314_t, domain_517_t)   # /sbin/runlevel

# Allowed process transition from /sbin/init:/etc/rc.d/rc:/etc/rc3.d/S85httpd
domain_trans(domain_115_t, file_13_t, domain_116_t)   # /bin/grep
domain_trans(domain_115_t, file_54_t, domain_117_t)   # /bin/touch
domain_trans(domain_115_t, file_62_t, domain_118_t)   # /sbin/consoletype
domain_trans(domain_115_t, file_63_t, domain_119_t)   # /sbin/initlog
```

そして、最後にドメインがアクセス可能なファイルやディレクトリに対して許可を与える。

```
# Allowed operations for /etc/rc3.d/S08ipchains
allow domain_547_t file_0_t:{ dir link_file } { r_dir_perms }; # /
```

```
allow domain_547_t file_11_t:{ dir link_file } { r_dir_perms }; # /bin/
allow domain_547_t file_15_t:{ file_class_set } { rx_file_perms }; # /bin/sed
allow domain_547_t file_83_t:{ file_class_set } { rx_file_perms }; # /bin/uname
allow domain_547_t file_16_t:{ dir link_file } { r_dir_perms }; # /dev/
allow domain_547_t file_17_t:{ file_class_set } { rw_file_perms }; # /dev/console
allow domain_547_t file_18_t:{ file_class_set } { rw_file_perms }; # /dev/null
allow domain_547_t file_1_t:{ dir link_file } { r_dir_perms }; # /etc/
allow domain_547_t file_55_t:{ dir link_file } { r_dir_perms }; # /etc/init.d/
allow domain_547_t file_56_t:{ file_class_set } { r_file_perms }; # /etc/init.d/functions
allow domain_547_t file_2_t:{ file_class_set } { r_file_perms }; # /etc/ld.so.cache
allow domain_547_t file_19_t:{ file_class_set } { r_file_perms }; # /etc/mtab
allow domain_547_t file_20_t:{ file_class_set } { r_file_perms }; # /etc/nsswitch.conf
allow domain_547_t file_21_t:{ file_class_set } { r_file_perms }; # /etc/passwd
allow domain_547_t file_75_t:{ dir link_file } { r_dir_perms }; # /etc/profile.d/
allow domain_547_t file_76_t:{ file_class_set } { r_file_perms }; # /etc/profile.d/lang.sh
```

6. 課題

カーネルでプロセス履歴を含むアクセス履歴を採取することにより、過不足のないポリシー定義情報を生成することが可能なことを確かめることができたが、最終的に定義されたアクセス情報からSELinuxのポリシー定義を生成する段階で、いくつか解決すべき課題が存在することに気がついた。

- アクセスを定義するためにはラベルを割り当てなければならない。今回の実装では、個々のファイルに対してラベルを付与しているため、必要なアクセス権を最小に抑えることの代償として、最終的なポリシー定義の量が膨大となってしまう。
- 本研究ではプロセスとドメインを一対一に対応させたが、実際には複数のプロセスをグループとしてまとめてドメインとすることによりポリシーの定義量を減らすことが望ましい。ただ、そのためには、プロセスの関係、意味内容の理解が必要であり、本論文で紹介したアクセス履歴情報を単にテキストとして処理する手法では実現できない。
- アクセス定義の対象としてポリシー定義の中でもっとも分量が多く、実運用の際にネックとなることが予想されるファイルに関するポリシーの生成を試みたが、/dev, /proc等の扱いは考慮しておらずその部分はSELinuxのデフォルトポリシーとマージする必要がある。

今後は、まずファイルアクセスについて/dev, /procへの対応を行い完結させ、次にファイルアクセス以外についても同じ手法を適用することにより必要十分なポリシーの自動生成を行えるようにする予定である。また、ポリシー生成機能についてはSubDomain等他の強制アクセス制御にも対応したいと考えている。

7. おわりに

SELinux開発者のメーリングリストでは、今も毎日のように個々のプログラムに対してあるべきポリシーの定義内容が議論されている。SELinuxを含めた強制アクセス制御はそれを使いこなすことにより高いセキュリティを得ることができるが、管理者にとっては制御できる機能の細かさとLinuxにおける動作スタイルが乗り越えがたい障壁となり、何らかの支援環境や指標なしにはそれを活用することができず、本来行うべき制限を定義できなかったり、逆に必要なアクセスを排除してしまったりすることによりサービスやプログラムが正しく動作できない状況が容易に起こりえる。

本論文で紹介した手法は、システム管理者に膨大な作業時間とスキルを求めることなく、目的とするサービスやアプリケーションを動作させ、それ以外の不要なアクセスを排除するポリシーを半自動的に得るための手段として有効であり、強制アクセス制御の運用上の課題を解決しその利点を最大限に引き出すものと位置づけられる。自動収集とアクセスポリシーの部分は特定の処理系に依存せずLinux/UNIXに汎用的に適用することが可能であり、本研究が不正アクセス対策の一助となることを願っている。

8. 参考文献

[1] P. Loscocco and S. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01)*, June 2001.

[2] National Security Agency, *Security-Enhanced Linux*, http://www.nsa.gov/selinux/

[3] Crispin Cowan et al, *SubDomain: Parsimonious Server Security*, 14th USENIX Systems Administration Conference (LISA 2000), December 2000.

[4] Amon Ott, *The Rule Set Based Access Control (RSBAC) Linux Kernel Security Extension*, International Linux Kongress 2001

[5] United States. Department of Defense, *TCSEC (Trusted Computer System Evaluation Criteria) DDS-2600-5502-87*, 1985

[6] Brian W. Kernighan and Rob Pike, *UNIX PROGRAMMING ENVIRONMENT*, 1985

[7] National Security Agency, *Flask: Flux Advanced Security Kernel*, http://www.cs.utah.edu/flux/fluke/html/flask.html

[8] 情報処理振興事業協会セキュリティセンター、「オペレーティングシステムのセキュリティ機能拡張の調査」, http://www.ipa.go.jp/security/fy13/report/secure_os/secure_os.html

[9] 「究極のセキュアOSを簡単導入」, Nikkei Linux 2003.5

[10] Stephen Smalley, *Configuring the SELinux Policy*, NAI Labs Report #02-007

[11] Chris Wright and Crispin Cowan et al, *Linux Security Module Framework*

9. 付録(カーネルから出力されたアクセスログの例)

(1) /bin/awk:x:/sbin/init:/etc/rc.d/rc.sysinit:/sbin/initlog:/etc/rc.d/rc.sysinit
(2) /bin/awk:x:/sbin/init:/etc/rc.d/rc:/etc/rc3.d/S10network
(3) /bin/awk:x:/sbin/init:/etc/rc.d/rc:/etc/rc3.d/S25netfs
(4) /etc/mtab:rw:/sbin/init:/etc/rc.d/rc.sysinit:/sbin/initlog:/etc/rc.d/rc.sysinit:/bin/mount
(5) /etc/mtab:r:/sbin/init:/etc/rc.d/rc:/etc/rc3.d/S25netfs:/sbin/initlog:/bin/mount

各項目はコロンで区切られている。最初の欄がアクセスされたファイルまたはディレクトリのフルパスである。2番目の欄がアクセスモード(Read/Write/Execute)である。3番目以降の欄はそのとき実行されていたプロセス遷移の履歴であり、最後の欄が実際にアクセス要求を発行したアプリケーションのフルパスである。

例えば(1)の履歴の場合、現在のプロセス(/etc/rc.d/rc.sysinit)は最初のプロセス(/sbin/init)が/etc/rc.d/rc.sysinitを起動し、/etc/rc.d/rc.sysinitが/sbin/initlogを起動し、/sbin/initlogが/etc/rc.d/rc.sysinitを起動することによって誕生したプロセスであるということが判る。この履歴全体(/sbin/init:/etc/rc.d/rc.sysinit:/sbin/initlog:/etc/rc.d/rc.sysinit)をドメインとして定義している。

(1)～(3)はいずれも/bin/awkを実行するためにファイルにアクセスしているが、それぞれ3番目の欄以降の内容が異なっているため、/bin/awkはそれぞれ

/sbin/init:/etc/rc.d/rc.sysinit:/sbin/initlog:/etc/rc.d/rc.sysinit:/bin/awk
/sbin/init:/etc/rc.d/rc:/etc/rc3.d/S10network:/bin/awk
/sbin/init:/etc/rc.d/rc:/etc/rc3.d/S25netfs:/bin/awk

という異なるドメインで実行される。

(4)と(5)は/bin/mountが/etc/mtabにアクセスするときの状況を示している。一般的な方法では/bin/mountに対して/etc/mtabの読み書き権限を与える必要がある。しかし履歴を使用することにより、/sbin/init:/etc/rc.d/rc.sysinit:/sbin/initlog:/etc/rc.d/rc.sysinit:/bin/mountのドメインからアクセスされた場合には読み書き権限を、/sbin/init:/etc/rc.d/rc:/etc/rc3.d/S25netfs:/sbin/initlog:/bin/mountというドメインでアクセスされた場合には読み込みのみを許可するというように、状況に応じてアクセス許可範囲を変更することが可能になる。