使いこなせて安全なLinuxを目指して

原田 季栄　保理江 高志　田中 一男
(株) NTTデータ オープンソース開発センタ
e-mail: {haradats, horietk, tanakakza}@nttdata.co.jp

概 要

Linuxカーネル2.6にはLSMとSELinuxが組み込まれ、「きめ細かなアクセス制御」を行うことが可能となった。だが、現実には適切なポリシーの作成が困難であるがために活用されているとは言えない。こうした状況の背景には、Linuxの構造とSELinuxの実装が深く関係している。筆者らは使いこなせて安全なLinuxを目指して、「改ざん防止Linux」および「TOMOYO Linux」を開発し、セキュリティ・スタジアム2004に「防御側」として出展、その効果を検証した。標準のLinuxにはどのような脅威があり、SELinuxはそれをどのように解決するか、何故SELinuxによる運用が困難となるかについて、独自に開発したLinuxの狙いや経緯、セキュリティ・スタジアムで得られた情報と合わせて報告し、今後のLinuxセキュリティ強化について考察する。

1. はじめに

筆者らはLinuxのセキュリティを改善するために、システムの大部分を読み込み専用メディアに記録することで物理的に改ざんを防止するSAKURA Linux[1]、およびシステムの実運用で課題となるセキュリティポリシーの自動学習機能を備えたTOMOYO Linux[2]を開発し、その実装についてLinux Conference （LC2003, LC2004）で発表した。2つのLinuxはそれぞれ、「ポリシーの管理運用を必要とせずに、安全確実に改ざんから守る」、「使いこなせる強制アクセス制御」を目指し開発した。現在ではSAKURAの内容はTOMOYO Linuxに統合され、起動時のオプションにより機能を選択することが可能となり、カーネル2.6にも対応した。2004年11月には、Linux Conference 2004以後追加したファイル以外の資源に対する強制アクセス制御機能を含めて、NPO 日本ネットワークセキュリティ協会が主催した「セキュリティ・スタジアム2004（SS2004）」[3]に防御側システムとして出展し、セキュリティ技術のエキスパートによる検証を受けた。

NSAが開発したSELinux[4][5]は、当初Linuxカーネルに対するパッチとして作成された。その後、Linuxカーネルを拡張するための汎用的なフレームワークであるLSM （Linux Security Modules）[6]とそれに対応した実装として書き直され、2004年12月よりカーネル2.6に正式に組み込まれた。これによりLinuxの利用者はコンパイルオプションのひとつとしてSELinuxを選択するだけで、その機能を利用できるようになった。

SELinuxとTOMOYO Linuxはアプローチと実装は異なるが、Linuxのセキュリティ強化を目指している点は共通である。本論文では、SELinuxとTOMOYO Linuxの概念やポリシー仕様の観点を中心としながら、Linuxセキュリティ強化の現状と課題について整理する。

2. OSセキュリティ強化の必要性

メールを読む、文書を作るなどのアプリケーションプログラムは、OSが提供する機能を組み合わせることにより実現されている。Linuxでは、OSの提供する機能はカーネル内にシステムコールとして用意されている。通常、アプリケーションは直接システムコールを用いて記述せず、libcをはじめとした共通的な機能を提供するライブラリを介してOSの機能を呼び出す。OSはアプリケーションやライブラリからの機能実行の要求を受けるとそれを実行するだけであり、個々の機能が全体としてどのようなサービスを構成するものか、また個々の処理の意味は判断することができない。そのため、バッファオーバーフロー等によりプログラムの制御を不正に奪われてしまっても、OSはそれを知ることがない。クラッカーの意のままに従順に指示に従い、システムに危害を与えることに協力してしまう。Linuxではあらゆる権限をrootに集中させるモデルに基づいており、管理者権限の奪取は致命的な問題につながる。

こうしたリスクを限定するためには、「不正な機能を判別し、その実行を制限する」ことが有効である。これを実

Towards a manageable Linux security.
Toshiharu HARADA, Takashi HORIE and Kazuo TANAKA,
Open Source Software Development Center, NTT DATA CORPORATION

現するものとして、1980年代より研究が行われた結果のひとつが強制アクセス制御 （MAC: Mandatory Access Control）である。強制アクセス制御については、DOD 5200.28-STD（TCSEC[7]）Division B: Mandatory Protection のClass B1: Labeled Security Protection等に記述されている。

3. 強制アクセス制御を適用する意味

強制アクセス制御はあらかじめ定義された正しいアクセスを判断するための条件であるポリシーに基づき、個々のアクセスを例外なく判断し正しい場合のみ実行を許可する。もともと汎用の用途に供するために開発されたOSに、特定の用途だけを実行させるための制限を加えることにより、システムの完全な乗っ取りを防ぎ、権限を奪われた場合の被害を限定、局所化することを可能とするものと理解することができる。しかし、強制アクセス制御はあくまで管理者が定めたルール（ポリシー）に従い、アクセスを制御するための「仕組み」であるから、それを有効に活用できるかどうかは適切なポリシーを書くことができるかによる。強制アクセス制御を導入することは、セキュリティを高めるための手段であり、それだけでセキュリティが高まるわけではない点に注意する必要がある。

4. 「物理的な改ざん防止」の効果とその限界

強制アクセス制御の有効性はTrustedOSと呼ばれる商用セキュリティ強化OSでも実証されているが、ポリシーの管理運用という代償を伴う。そのため、例えば情報を公開するのが目的のサーバや個人が立ちあげているホームページ等で、強制アクセス制御を実装したOSを使うには負担が大きい。そこで筆者らはポリシーの管理運用という負担を課すことなく、それらのシステムを保護できないかと考え、2003年にシステムの大部分を読み込み専用メディアに記録することで物理的に改ざんを防止するSAKURA Linuxを開発、その実装を発表した[1]。SAKURA Linuxでは、カーネル2.4に対し、次の拡張を行っている。

- root fsの内容を物理的に書き換えから保護できるメディア（DVD-R, USBフラッシュメモリ等）に保持する。
- mount, chroot, pivot_root等の実行を制限する。
- /tmp等書き込みが必要なパーティションに対して、noexecオプションを強制する。

SAKURA Linuxでは、SELinuxのようなポリシーファイルは存在しない。しかし、ファイルシステムの分類自体が意味的に暗黙の（非常に粗い）ポリシーに相当し、物理的に書き換えを防ぐという強制アクセス制御が実装されていると考えることができる。改ざん防止自体は、物理的なので完全だが、root fsの上にマウントされてしまっては保護の意味がなくなる。その部分についてはカーネルを一部修正した。SAKURA Linuxでは、root権限を奪われてもカーネルやコマンド、共有ライブラリおよび静的なwebコンテンツは物理的に保護されており、改ざんの被害を受けない。細かなアクセス制御を指定することはできないが、用途と効果を限定することによりポリシーの運用の負担を不要としている点が特徴である。SAKURA LinuxについてSS2004に防御側として出展し、その有効性を検証した。その結果、直接改ざんの被害を受けることはなかったが、以下の攻撃を受けた。

- Sambaサーバの脆弱性を突かれて、システム管理者権限を持つシェルを起動された。
- 起動していたhttpdとsshdを停止された。
- 偽物のhttpdを実行された。
- /devディレクトリのデバイスファイルを削除された。

SAKURA Linuxでは、SELinuxにおけるようなアクセス制御は実装しておらず、mountに関する制限以外はほぼ標準のLinuxのままである。従って、root権限を奪われると、改ざんは受けなくてもシステムの再起動を含めた「管理者の操作」に対しては、標準のLinuxと同様これを実行してしまう。/tmp等の書き込みを許したパーティションで、バイナリプログラムを実行させることは防いでいるが、Javaで書かれた小さなhttpサーバをコンパイルされ、実行されてしまった。その結果、「改ざんはされていないが外部から見たときに意図していないwebコンテンツを提供する」状態が実現してしまった。同様のことは、Perl, Ruby等のインタプリタが搭載されている場合にも起こり得て、これは物理的な改ざん防止だけでは防ぐことはできない。また、/devディレクトリのデバイスファイルを削除されたことにより、正規の管理者がコンソールからログインすることが不可能となってしまった。強制アクセス制御に限らず、ソフトウェアで実現するものについては、バグや脆弱性はつきまとう。その意味において、物理的な改ざん防止の有効性に対する筆者らの考え方は今も変わってはいない。しかし、SS2004の結果を通じて、少なくともセキュリティを重視する用途へのLinuxの適用は、物理的な改ざん防止だけでは十分ではなく、強制アクセス制御の導入が「前提」となるという認識を新たにした。

5. SELinux

Linuxにおける強制アクセス制御の実装としてもっとも知られているものは、NSAが開発、公開したSELinuxであろう。SELinuxは、LinuxにTE（Type Enforcement）とRBAC（Role-Based Access Control）のセキュリティモデルを実装し、「柔軟で粒度の細かいアクセス制御」を可能としたものである。アクセス制御の汎用的なフレームワークであるLSMに対応した形でLinuxカーネル2.6に組み込まれ、導入の障壁は解消した。既存のプログラムとの互換性を残し、パフォーマンス面での影響を抑えながらきめのこまかなアクセス制御を可能としたが、「いかにして適切なポリシーを書くか」という大きな課題が解決されず残っている。

5.1. SELinux利用の実際

2005年3月、米国メリーランドで開催されたSELinux Symposium[8]では、「いかにして適切なポリシーを得るか」が議論のひとつの焦点となった。ディストリビューターの中でいち早くSELinuxをデフォルト有効な状態で組み込むなど、SELinux統合に積極的なRed Hat社からは「ポリシーを理解できない人がポリシーを書くべきではない」という意見も示された。実際、Fedora Core 3では、SELinuxをオンにすると”targeted”と呼ばれる「部分的な」防御が有効となるが、そのtargetedのポリシーのソースファイル（定義ファイル）は独立したRPMパッケージであり、管理者が陽に追加しなければ参照することもできないようになっている。

SELinuxには、”default policy”と呼ばれる実装と合わせて配布されているポリシーが存在する。default policyは、約3万行のテキストファイルがその定義実体である。SELinuxの本来あるべき利用は、このdefault policyを完全に理解した上で、ドメインの定義（構成）を変更し、それぞれのドメインにおけるアクセスを定義することである。しかし、そのためにはOSとアプリケーションの振る舞いを詳しく正確に把握していることが前提となる[9][10][11]。残念ながらそれは誰もが容易になしとげられる作業ではない。細かな制御を可能にするためのセキュリティ強化OSが、default policyやそれに対する微修正により使われているというのが現状である。SELinuxにおけるアクセス制御の概念を整理することから始めて、この状況をもたらしている原因について考えてみたい。

5.2. SELinuxにおける概念

SELinuxでは、「ドメイン」、「タイプ」、「コンテキスト」、「ラベル」、「role」という概念に基づきアクセス制御を行う[10]。試みに、それらを極力短く定義してみると以下のようになる。

『SELinuxでは、ファイル、ディレクトリ、ソケット等のオブジェクト（「オブジェクトクラス」）について与えるべきアクセス権限（Access Vector）をきめこまかく定義できる。与える権限は、単一固定したものではなくプロセスとプロセスがアクセスしようとしている対象の組み合わせ等の「状況」により異なるのが通常である。SELinuxではこの「状況」を区別し、「状況」毎に異なる権限を規定できるようにするために「コンテキスト（セキュリティコンテキスト）」と呼ばれる概念を導入している。「コンテキスト」は、ユーザ、ユーザが属する役割（role）、オブジェクトに付与された「タイプ」と呼ばれる名札により構成されている。コンテキストのうち特にプロセスにつけられた「タイプ」をドメイン、それ以外につけられた「タイプ」をラベルと呼ぶ。「ドメイン」はプロセスの側から見たときには、自らが属するタイプであるが、実際にはアクセスを制御したい範囲毎に定義するものであり、その意味では「アクセス許可定義の集合体」である。従って、ドメインがまず定義されていて、それが必要に応じてプロセスに割り当てられると考えると良い。コンテキストにおける"role"は、「役割」を意味する。SELinuxが導入されたシステムでは、ユーザは必ず何かのroleに属するが（id -Z、あるいはid --contextにより確認できる。）、複数のroleを割り当てるようポリシーを記述することもできる。roleは、前述のドメインの集合体（セット）と考えることができ、ユーザは自分の"role"に割り当てられているドメインの中から、newroleコマンドを実行することにより好きなドメインを選択し、作業を行うことができる。』

5.3. 何故SELinuxのポリシー策定は困難なのか

SELinuxのポリシーを書く上での最初の関門は、前述のSELinuxの概念を理解することであるが、それだけではない。筆者らは、SELinuxのポリシー策定が困難な理由について、次の3点の影響が大きいと考えている。

- アクセス制御（ポリシー記述）の粒度が細かすぎるため、システムの挙動を把握することが困難である。例えば、ファイルに対するパーミッションとして、”r_file_perms”など13種が定義されている。（しかもその13種はSELinuxにおけるプリミティブではなく、設定をわかりやすくするためのマクロである。）
- ドメインの定義内容が管理者の裁量に委ねられている。SELinuxでは、ドメイン毎に許可するアクセスを定義する。ドメインを分割することにより細かく許可を与え、セキュリティを高めることができるが、どの単位で

ドメインを構成するかは基準がない。

- アクセス制御をファイル名やディレクトリ名ではなく、ラベルに基づき指定しなければならない。カーネルの中では、ファイルやディレクトリはパス名ではなく、iノードにより管理されている。SELinuxにおけるファイルやディレクトリに対するアクセス制御は、iノードに付与されたタイプ、すなわちラベルにより行う。そのため、ラベルの定義（パス名とラベルの対応付け）を行い、それを維持する必要がある。

6. ポリシーの自動学習とその課題

Linuxのセキュリティ強化の鍵となるポリシーについて、「人間が把握、記述することが困難であるのならば、それを自動的に定義できないだろうか」と考え、筆者らが開発したのが、Network Security Forum 2003 (NSF2003) で発表した「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」[12]である。基本的な発想は単純で、「カーネルでアクセス制御をさせるというのであれば、カーネルでアクセス要求を観測、記憶しておけば、強制アクセス制御のポリシーを生成できるのではないか」ということだった。NSF2003では、/sbin/initに始まるプロセス履歴のツリーをドメインの単位として観測し、その情報をもとにSubDomain風のポリシー定義の生成が可能なことを示したが、取り組みのきっかけとなったSELinuxのポリシーについては生成に至っていない。

6.1. ドメイン定義

SELinuxでは、前述のようにフラットで階層を持たない「ドメイン」毎にポリシーを記述し、指定したプログラムの実行によりドメイン間の遷移を定義するよう実装されている。そのため、もし仮にポリシーを記述しなければならない対象システムの挙動を完全に把握していたとしても、それをどのようなドメイン (群) に分割するかは一意には定まらない。どのような単位で分割するか分からないものについてそれを自動的に定義することはできない。MITREの開発しているpolgen[13]では、straceを用いて、ポリシー定義の自動化を行おうとしているが、straceから得られる情報をどのようにSELinuxのドメインと対応させるかは、管理者の判断と作業に委ねられており、あくまでポリシー定義の参考情報を生成するにすぎない。筆者らがNSF2003で発表した方式では、pstreeの出力結果のように、実行されるプロセスについてその起動 (exec) の履歴を文字列として結合したものをドメインとするという方式をとった。例えば、同じ/bin/mountであっても、図1の例のように実行された履歴により区別し、それぞれのコンテキストでアクセスを観測する。

```
/sbin/init /etc/rc.d/rc.sysinit /sbin/initlog /bin/mount
（initから実行されたrc.sysinitから実行されたinitlogから実行されたmountプログラムのドメイン）
/sbin/init /etc/rc.d/rc /etc/rc3.d/S25netfs /sbin/initlog /bin/mount
（initから実行されたrcから実行されたS25netfsから実行されたinitlogから実行されたmountプログラムのドメイン）
```

図1. 「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」におけるドメインの定義例

「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」におけるドメインの分割は、同じことをSELinuxで行おうとした場合、全てのプログラム実行を契機としてドメイン遷移条件を定義することとみなすことができる。それはポリシーの文法上は可能であるが、SELinuxが備えるアクセス制御の粒度を考えると、管理者が定義することは非現実的であり、できたとしてもポリシー定義量の肥大化によりパフォーマンスへの影響が大きくなることが予想される。

6.2. 定義の粒度

SELinuxにおけるアクセス許可の粒度は、そのベースとなっているフレームワークLSMの粒度がベースとなっている。LSMはカーネルの内部でシステムコールの呼び出しをフックするようになっているので、アクセス許可はカーネル内部の実装に合わせて記述する必要がある。ユーザランドのアプリケーションを開発する際にはカーネル内部の実装を考慮しながら実装する必要は無いのにもかかわらず、ポリシーの定義はカーネル内部の実装を考慮して記述しなければならないという矛盾が生じている。これは現在のLSMにより強制アクセス制御を行う限り逃れられない制約となる。

TOMOYO Linuxでは、LSMによらずに独自にいくつかのシステムコールにパッチを当てることにより、ポリシーの自動学習を行い、その対称としての強制アクセス制御を可能とした。ポリシーの粒度については、標準的なLinux/UNIXのシステム管理者が理解、把握できることを要件として考え、read/write/executeの3種として、それをビット論理和として記述するようポリシー言語仕様を定めた。TOMOYO Linuxのポリシー表記はLinuxの利用者であれば誰でも一目でその内容を理解できるし、編集することも容易である。

6.3. ラベル

TOMOYO Linuxでは、直感的でわかりやすいポリシーの運用管理を行うためには、ポリシー中でファイルやディレクトリ名称をそのまま記述できることが必須であると考えた。ポリシーの学習およびアクセス制御の実施の際に、要求されたパス名を、シンボリックリンク等を含まない絶対パスに変換する処理を追加することにより、ポリシーファイルでパス名を使った表記に対応することができた。(具体的な実装については文献[2]を参照されたい。)

7. TOMOYO Linuxのポリシー

セキュリティを強化されたLinuxのふるまいはポリシーにより定義される。そのため、ポリシーの仕様はセキュリティを強化されたLinuxの機能の写しであり、ポリシーの可読性はそのままセキュリティ強化OSの運用性の尺度となる。本章では、2005年4月現在のTOMOYO Linuxについてポリシー仕様を紹介する。これらの内容を理解することができる人であれば、TOMOYO Linuxを使いこなすことができる。ポリシーの仕様の一部は、LC2004発表後、TOMOYO Linuxを実際のシステムに適用する過程で必要性を感じて追加したものが含まれている。

7.1. TOMOYO Linuxの概念

SELinuxと同様にTOMOYO Linuxにおける概念を以下に説明してみる。

『プロセスの実行(exec)ツリー毎にドメインを定義し、ドメイン毎にそれがアクセスする対象をファイル、ディレクトリ名そのままにread, write, executeの粒度で制御する。ドメインの分割および必要なアクセスの学習はカーネルの起動時のモード指定により自動的に行うことができる。』

7.2. ドメイン定義

TOMOYO Linuxでは、ドメインに対してアクセス許可を与えるという形式を採用している。既定のドメインとして、カーネルが属する「<kernel>」を定め、それ以降プロセスの起動履歴を並べることにより、ドメインを定義する。例えば、/sbin/initはカーネルから起動されるので、/sbin/initが属するドメインは「<kernel> /sbin/init」となり、また、その状態から/etc/rc.d/rcが起動されるので、(カーネルから起動された/sbin/initが起動した)/etc/rc.d/rcの属するドメインは「<kernel> /sbin/init /etc/rc.d/rc」と表現される。パス名はすべてシンボリックリンクや .. などを解決した絶対パスで表現される。この方法により全てのプロセスが属するドメインを一意的に定めることができる。

7.3. ポリシーファイルの構成

TOMOYO Linuxでは、以下のポリシーファイルを使用する。常に全てのポリシーファイルを用意する必要は無く、強制アクセス制御を有効にしたい機能に対応したポリシーファイルだけ作成すればよい。

(1) policy.txt
全てのドメインを定義し、ファイルに対するアクセス許可を定義する。7.4を参照。

(2) allow_bind.txt
ドメインがbindすることができるローカルポート番号を定義する。7.5を参照。

(3) cap_policy.txt
ドメインが使用できるケイパビリティを定義する。7.6を参照。

(4) authorized.txt
上記3種類の強制アクセス制御を受けないドメインを定義する。7.7を参照。

(5) initializer.txt
ドメイン遷移履歴をリセットするプログラムを定義する。7.8を参照。

(6) allow_read.txt
全てのドメインが読むことができるファイルを定義する。

(7) chroot.txt
chrootで移動可能なディレクトリを定義する。

(8) mount.txt
マウントを許可するパーティション・マウントポイント・ファイルシステム・オプションの組み合わせを定義する。

(9) noumount.txt
アンマウントを許可しないマウントポイントを定義する。

(10) syaoran.conf

作成を許可するデバイスファイルのファイル名と属性の組み合わせを定義する。7.10を参照。

7.4. ファイルに対するアクセス制御

ファイルに対するアクセス許可は、ドメイン定義の行に続けて、アクセスモード(数値化したパーミッション)とパス名を指定する。あるドメイン定義行から次のドメイン定義の直前の行までがそのドメインに与えられるアクセス許可である。ファイルの種別はディレクトリか否かのみである。ディレクトリは"/"で終わり、ディレクトリ以外は"/"で終わらない。読み書きの許可に関してはワイルドカードを使用可能である。図2にアクセスポリシーの記述例を示す。

```
<kernel>
1 /sbin/init
<kernel> /sbin/init
6 /dev/console
6 /dev/initctl
6 /dev/tty¥$
4 /etc/inittab
6 /etc/ioctl.save
4 /etc/localtime
1 /etc/rc.d/rc
1 /etc/rc.d/rc.sysinit
1 /sbin/mingetty
1 /sbin/shutdown
2 /var/log/wtmp
6 /var/run/utmp
```

<kernel> は以下のことができる。
- /sbin/init を実行 (--x)

<kernel> から起動された /sbin/init は以下のことができる。
- /dev/console の読み書き (rw-)
- /dev/initctl の読み書き (rw-)
- /dev/tty[0-9]* の読み書き (rw-)
- /etc/inittab の読み込み (r--)
- /etc/ioctl.save の読み書き (rw-)
- /etc/localtime の読み込み (r--)
- /etc/rc.d/rc を実行 (--x)
- /etc/rc.d/rc.sysinit を実行 (--x)
- /sbin/mingetty を実行 (--x)
- /sbin/shutdown を実行 (--x)
- /var/log/wtmp の書き込み (-w-)
- /var/run/utmp の読み書き (rw-)

図2. ファイルのアクセス許可の定義例 (policy.txt)

実行許可を含む場合はワイルドカードを使用できないようになっている。これは、以下の理由による。
- ワイルドカードが含まれていると、予期せぬドメイン遷移が発生することがある。
- 1つの実行許可から複数のドメインに遷移できると、エディタでの表示が困難になる。
- 管理者がエディタを使用してドメイン遷移を確認する場合に、見落としが発生しやすくなる。

プログラムの実行許可の判断時に読み込み許可のチェックは行っていない。これは、以下の理由による。
- 任意アクセス制御において、読み込み許可のチェックが行われている。
- プログラムの内容を必要以上にユーザに見せたくない。

7.5. ローカルポートに対するアクセス制御

TCPおよびUDPを使用するプログラムに対して、使用できるローカルホスト側のポート番号を制限できる。

図3にポリシー記述例を示す。ポリシーはドメイン毎にプロトコルとそのドメインに対してbind操作を許すポートの番号を<プロトコル>-<ポート番号>の形式で記述する。ポート番号に0が書かれている場合は、任意のポートを示す。ドメイン毎に複数のポートを指定できる。本機能はLC2004発表後実装した。

```
<kernel> /sbin/portmap
UDP-0 TCP-0
<kernel> /usr/sbin/dhcpd
UDP-67
<kernel> /usr/sbin/httpd
TCP-443 TCP-80
```

・portmapはUDPとTCPの任意のポート番号にbindできる。

・dhcpdはUDPの67をbindできる。

・httpdはTCPの443と80をbindできる。

図3. bind許可の定義例 (allow_bind.txt)

7.6. ケイパビリティに対するアクセス制御

LC2004発表後、下記のシステムコールに対するアクセス制御を実装した。

- socket(PF_INET or PF_INET6, SOCK_STREAM, *), socket(PF_INET, SOCK_DGRAM, *), socket(PF_INET, SOCK_RAW, *), socket(PF_ROUTE, *, *), socket(PF_PACKET, *, *)
- listen() for PF_INET or PF_INET6, SOCK_STREAM
- connect() for PF_INET or PF_INET6, SOCK_STREAM
- sys_mount(), sys_umount(), sys_reboot(), sys_chroot(), sys_kill(), sys_tkill(), sys_vhangup(), do_settimeofday(), sys_adjtimex(), sys_nice(), sys_setpriority(), sys_sethostname(), sys_setdomainname()

TOMOYO LinuxではLinux標準のケイパビリティである「CAP_*」には手を加えなかった。その理由は、ケイパビリティの種類が少ないため、1つのケイパビリティ（例えばCAP_SYS_ADMIN）が多数の用途でチェックされており、システムコール単位で制限を課すには粒度が粗すぎるためである。TOMOYO Linux独自のチェックは、Linux標準のケイパビリティによるチェックとは独立に行われる。図4にポリシー記述例を示す。ドメインの指定については、ファイルに対するアクセス制御の記法と同一である。

```
<kernel> /sbin/mingetty /bin/login /bin/tcsh /usr/bin/ssh
inet_tcp_create inet_tcp_connect use_inet_udp
<kernel> /sbin/portmap
inet_tcp_create inet_tcp_listen use_inet_udp
<kernel> /usr/sbin/httpd
inet_tcp_create inet_tcp_listen use_route SYS_KILL
```

図4. ケイパビリティの定義例 （capability.txt）

7.7. 信頼済みドメイン

ポリシーの開発の過程においては、強制アクセス制御を有効にしたままで、システム管理者として、コマンドの実行、ファイルの編集等を行いたい場合が存在する。そのような場合、いちいちアクセス制御を無効にしたり、あるいは管理用のポリシーを追加するのでは作業効率が悪い。そこで、強制アクセス制御が有効な状態にあっても、Linux標準の任意アクセス制御しか受けないドメインを定義できるようにした。例えば、sshdから起動されたシェルに対して、アクセス制御を受けないように指定することにより効率的にポリシーの編集が行える。図5に信頼済みドメインの定義例を示す。

```
<kernel> /usr/sbin/sshd /bin/tcsh
<kernel> /sbin/mingetty /bin/tcsh
```

・/usr/sbin/sshdから起動された/bin/tcshについては信頼する。
・/sbin/mingettyから起動された/bin/tcshについては信頼する。

図5. 信頼済みドメインの定義例 （authorized.txt）

信頼済みドメインの典型的な用途のひとつにパッケージ管理がある。例えば、rpmのドメインを信頼済みドメインとするような使い方を想定しているが、信頼済みドメインは強制アクセス制御の例外となるため使用に際しては細心の注意が必要である。本機能は、TOMOYO Linuxを実際のシステムに適用した際の要望を機能として実装したものである。

7.8. ドメイン遷移例外

システムのメンテナンス時には、httpd等通常は自動起動するようになっているプログラムを、ssh等により別のドメインから再起動させたい場合がある。そのような場合、通常であればTOMOYO Linuxは同じhttpdプログラムであっても全く別のドメインとして扱うが、これを同一にできたほうが望ましい。そこで、指定されたプログラムが実行された場合、必ず「<kernel>」直下のドメインに遷移させる「ドメイン遷移例外」機能を実装した。本機能は、複数のプロセス起動履歴を集約するための機能と言える。図7に /usr/sbin/httpd が実行された場合のポリシー記述例を示す。（プログラムファイル以外は省略している。） 本機能もまた実際にTOMOYO Linuxを適用した際のニーズに応えて実装したものである。

```
/usr/sbin/httpd
```

図6. ドメイン遷移例外の定義例 （initializer.txt）

```
<kernel> /init /sbin/init /etc/rc.d/rc /etc/rc.d/init.d/httpd /sbin/initlog
1 /usr/sbin/httpd
<kernel> /sbin/mingetty /bin/login /bin/tcsh /sbin/service /bin/env /etc/rc.d/init.d/httpd
/sbin/initlog
1 /usr/sbin/httpd
<kernel> /usr/sbin/httpd
1 /var/www/cgi-bin/sessioncookie.cgi
```

図7. ドメイン遷移例外を使用した場合のファイルアクセス許可の定義例 (policy.txt)

7.9. イベントドリブンドメイン

TOMOYO Linuxのポリシー自動学習機能の評価を行った際に、/sbin/hotplugや/sbin/modprobeなどのプログラムが実行されるドメインが変動する場合があることに気がついた。これらのプログラムはシステムの動作状況により実行されたり実行されなかったりするため、自動学習したポリシーだけで強制アクセス制御をかけると必要な機能が実行できないことがある。そこで、カーネルがこれらのプログラムを起動するためにカレントプロセスをforkした時点で強制的に「<kernel>」ドメインに割り当てるようにした。これによりシステムの動作状況に依存するプログラムの自動学習が可能となり、システムの運用が容易となった。

7.10.デバイスファイルの保護

LC2004発表時点では、TOMOYO Linuxはファイルの種別は考慮せず、ファイル名による強制アクセス制御を行うよう実装していた。そのため、あるファイル(名)に対する書き込み権限を持っていれば、そのファイルを削除して同じ名前のファイルを作成することができてしまう。もし作成するファイルがデバイスファイルであった場合には、ハードディスクの内容を一度に破壊することも可能となる。この問題を回避するために、tmpfsをベースとして以下の機能を持つ独自のファイルシステムを実装した。

- 作成されるファイルの名前とその種類(ディレクトリ、FIFO、ソケット、キャラクタ型デバイス、ブロック型デバイス、シンボリックリンクのいずれか。キャラクタ型デバイスまたはブロック型デバイスの場合はメジャー番号およびマイナー番号を含む。)を制限できる。
- ファイルの削除や属性(所有者・グループおよびパーミッション)の変更の可否をポリシーにより制御できる。
- 学習モードを備えており、アクセスされたデバイスファイルのみを抽出できる。

このファイルシステムを/devにマウントすることにより、例えば/dev/nullがメジャー番号1、マイナー番号3のキャラクタ型デバイスファイルであることを保証できる。また、/dev/nullの所有者やグループおよびパーミッションの変更を禁止することができる。本手法についてもっと知りたい方は文献[14]を参照されたい。

8. TOMOYO Linuxポリシーの応用

TOMOYO Linuxのドメイン定義規則を活用することで、以下のような応用が可能になる。

8.1. ログイン認証の強化

システムにログインするにはユーザ認証が行われているが、パスワードを使用している場合、辞書攻撃等により突破される危険性が存在する。また、認証を行うプログラムの脆弱性により認証を回避される可能性も存在する。強制アクセス制御は、単純かつ効果的にこれらの問題を改善することができる。ここでは、TOMOYO Linux のポリシーを例に、カスタマイズ可能なログイン認証の強化について紹介する。

図8にアクセスポリシーの定義例を示す。また、簡単な認証プログラムの例を図9に示す。

```
<kernel> /usr/sbin/sshd /bin/bash
1 /bin/auth1
<kernel> /usr/sbin/sshd /bin/bash /bin/auth1
1 /bin/bash
<kernel> /usr/sbin/sshd /bin/bash /bin/auth1 /bin/bash
1 /bin/auth2
```

```
<kernel> /usr/sbin/sshd /bin/bash /bin/auth1 /bin/bash /bin/auth2
1 /bin/bash
<kernel> /usr/sbin/sshd /bin/bash /bin/auth1 /bin/bash /bin/auth2 /bin/bash
1 /bin/cat
1 /bin/vi
```

図8. 追加のログイン認証を行うためのポリシーの定義例 (policy.txt)

```
#! /bin/sh
for i in 1 2 3
do
  read -r -s -p 'Password: ' passwd
  echo
  [ "$passwd" = "SAKURA" ] && exec $SHELL
done
echo 'Incorrect password.'
```

図9. 単純な認証プログラムの例

```
#! /bin/sh
for i in 1 2 3
do
  read -r -s -p 'Password: ' passwd
  echo
  [ -f /data/rootauth ] && exec $SHELL
done
echo 'Incorrect password.'
```

図10. パスワードに拠らない認証プログラムの例

/bin/auth1 と /bin/auth2 というのが追加された認証を行うためのプログラムである。最初はsshログインにより起動される/bin/bashのアクセス許可であり、/bin/auth1の実行だけが許可されている。/bin/auth1の役割は、認証を行い、次のプログラム(/bin/bash)を起動することである。そのため、/bin/auth1は/bin/bashを実行する許可だけあればよい。同様に、/bin/auth2における認証に成功後/bin/bashが起動され、初めて/bin/catや/bin/viの実行(実質的な操作)が行えるようになる。時刻情報を利用してパスワードが変化するRSA社のSecurIDのような認証プログラムも可能である。認証に使える情報はパスワードだけに限定されないため、図10に示すように特定のファイルが存在するかどうかや、ファイルの内容をパスワードとして利用することもできる。通常のアプリケーションと同じ感覚で作成でき、実現できる組み合わせは無限である。本手法についてもっと知りたい方は文献[15]を参照されたい。

8.2. RBAC

SELinuxのようにRBAC(Role-Based Access Control)を実装したシステムでは、"role"を適切に定義することで権限分割を行うことが可能である。しかし、要素が増える分、ポリシーの記述は複雑になる。TOMOYO LinuxはRBACの機能を持たないが、簡単な工夫によりRBAC風の権限分割を行うことが可能である。基本的な考え方は、8.1と同じである。認証プログラムの代わりに、任意のプログラム(例えばシェル)を実行させることでドメイン遷移を分割し、それぞれのドメインに必要最小限のアクセス許可を付与することでRBAC風の権限分割を実現している。図11にアクセスポリシーの定義例を示す。

```
<kernel> /usr/sbin/sshd /bin/bash
1 /bin/bash
1 /bin/tcsh
1 /bin/zsh
<kernel> /usr/sbin/sshd /bin/bash /bin/bash
 (WWWサーバの管理に必要なアクセス許可のみを記述)
<kernel> /usr/sbin/sshd /bin/bash /bin/tcsh
 (メールサーバの管理に必要なアクセス許可のみを記述)
<kernel> /usr/sbin/sshd /bin/bash /bin/zsh
 (アカウントの管理に必要なアクセス許可のみを記述)
```

図11. システム管理者権限を分割するためのポリシーの定義例 (policy.txt)

最初はsshログインにより起動される/bin/bashのアクセス許可であり、そこでは/bin/bashと/bin/tcshと/bin/zshの実行だけが許可されている。ここで、/bin/bashを実行すると、「<kernel> /usr/sbin/sshd /bin/bash /bin/bash」というドメインに遷移

するので、これを基点としてWWWサーバの管理に必要なアクセス許可を記述する。もちろん、手作業で定義する必要は無く、TOMOYO Linuxを学習モードで実行させるだけでよい。また、/bin/tcshを実行すると、「<kernel> /usr/sbin/sshd /bin/bash /bin/tcsh」というドメインに遷移するので、これを基点としてメールサーバの管理に必要なアクセス許可を記述する。/bin/zshを起動した場合も同様である。本手法についてもっと知りたい方は文献[16]を参照されたい。

9. おわりに

Linuxに限らずOSのセキュリティ強化は、本来汎用の用途に供するために作られたOSに、その汎用性を後から制限することを目的としている。後付けであるが故に、そこには無理が生じ、それはポリシーの記述に顕れる。SELinuxはよく「柔軟で粒度の細かい強制アクセス制御を実装した」として言及される。それはやや偏った見方であり、実際には実装上の都合により、「ポリシーをシステムコールのレベルで記述しなければならない」という見方もできるだろう。高級言語で実現できる事は、原理的にはアセンブラでも実現できないことはない。それと同じように、Linuxのセキュリティは SELinuxのポリシーで実現可能であろうが、それを実用の領域に持ち込むためには、本論文で指摘した課題に関して、今後いくつかのブレークスルーが必要であろう。例えば、ポリシー違反が発生した場合に、それをポリシーとして追加することによる影響を解析したり、既存のドメインを分割する際の処理の自動化等があったりすると状況は大きく改善される。パッケージ管理にそのパッケージが必要とするアクセス許可情報を盛り込み、調停することができれば、管理者の負担なく、適切なポリシーの調整ができるかもしれない。

SELinuxを、「あるべきアクセス制御」を整然とLinuxの上に実現したものとすれば、筆者らが開発したTOMOYO Linuxは、最初から「使いやすさ」を意識し、運用可能な状態を維持しながらOSに余計なことをさせないように模索した結果である。TOMOYO Linuxは現在ソースコードの公開に向けて、作業を進めている。TOMOYO Linuxの使いやすさとわかりやすさ、あるいはその実装が、SELinuxや他のセキュリティ強化OSにとって改善のヒントとなることを、Linuxや他のOSが安心して使いこなせる日がくることを願ってやまない。

謝辞 TOMOYO Linux開発については企画段階から実装まで、NTTデータカスタマサービスの半田哲夫氏に支援いただきました。半田氏に深く感謝します。

参考文献

[1] 原田季栄、保理江高志、田中一男 「読み込み専用マウントによる改ざん防止Linuxサーバの構築」 Linux Conference 2003, http://lc.linux.or.jp/lc2003/30.html
[2] 原田季栄、保理江高志、田中一男 「TOMOYO Linux – タスク構造体の拡張によるセキュリティ強化Linux」 Linux Conference 2004, http://lc.linux.or.jp/lc2004/03.html
[3] http://www.jnsa.org/seminar_20041101.html
[4] http://www.nsa.gov/selinux/
[5] Fedora Core 3 SELinux FAQ, http://fedora.redhat.com/docs/selinux-faq-fc3/index.html
[6] http://lsm.immunix.org/
[7] Department of Defense, *Trusted Computer System Evaluation Criteria*, December 1985
[8] SELinux Symposium, http://www.selinux-symposium.org/
[9] Stephen Smalley and Timothy Fraser. *A Security Policy Configuration for the Security-Enhanced Linux*, February 2001
[10] Stephen Smalley, *Configuring the SELinux Policy*, February 2005.
[11] Michelle J. Gosselin, Jennifer Schommer, *Confining the Apache We Server with Security-Enhanced Linux*
[12] 原田季栄、保理江高志、田中一男 「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」 Network Security Forum 2003, http://www.jnsa.org/award/2003/result.html
[13] MITRE – Security-Enhanced Linux, http://www.mitre.org/tech/selinux/
[14] 原田季栄、保理江高志、田中一男 「SYAORAN – デバイスファイルの正しさを保証するファイルシステム」
[15] 原田季栄、保理江高志、田中一男 「CERBERUS - 強制アクセス制御を利用したログイン認証の多重化手法」
[16] 原田季栄、保理江高志、田中一男 「YUE - TOMOYO Linux向け管理者権限分割手法」