Vol.2011-DPS-146 No.43
Vol.2011-CSEC-52 No.43
2011/3/11

アプリケーションの実行を指定した時間に制限するためのLinuxの拡張

石黒　駿[†1]　　大山　恵弘[†1]

作業効率やセキュリティの観点から，特定のユーザが特定のアプリケーションを実行することを管理者が制限したい場合がある．たとえば，一部のユーザにはウェブブラウザやゲームを利用させたくない場合がある．このような実行制限は，多くの場合，勤務時間内などの特定の時間内にのみ必要である．もし，指定した時間内にのみ実行が制限されるならば，ユーザは休み時間中にそれらのアプリケーションを楽しむことができる．そこで本研究では，管理者がアプリケーションおよびそれらのアプリケーションの実行が許される時間を指定することができるシステムを提案する．もし実行が許されない時間にアプリケーションが起動されると，その実行が拒否されるか，警告が表示される．本システムを Linux の拡張として実装し，実験を通じて評価した．

A Linux Extension to Impose Time-Based Restriction on Executions of Applications

Shun Ishiguro[†1] and Yoshihiro Oyama[†1]

An administrator of a computer sometimes would like to impose restrictions on execution of particular applications by particular users from a viewpoint of working efficiency and security. For example, some administrators would like some users not to execute Web browsers or games. In many cases, these execution restrictions are needed only during specific hours such as working hours. If the executions are restricted only during a specified period of time, the users can enjoy the applications during a rest. In this work, we propose a system that allows administrators to specify the applications and periods of time during which the applications are allowed to run. If a specified application is invoked while the execution is not allowed, the system blocks the execution or shows a warning. We implemented the system as an extension to Linux and evaluated it through experiments.

1. はじめに

コンピュータを利用する上で，プログラムの実行を制限したい場合がある．たとえば，家庭や学校等で子供がパソコンを利用する場合に，そのパソコンで実行することが可能なアプリケーションを制限したいという場面が挙げられる．また，企業等でも，社員や職員の統制を行うという目的で，同様の制限を行いたい場面がある．

アプリケーションの実行を不可能にするために，複数の方法が存在する．第一にアプリケーションそのものをアンインストールしてしまうという方法が考えられる．第二に，Linux のパーミッションやセキュア OS の強制アクセス制御[1)2)] などの既存のアクセス制御機構を利用して，アプリケーションの実行ファイルに対して，実行制限をかける方法がある．しかし，これらの方法では，アプリケーションの実行が制限される時間を限定し，勤務時間中は禁止したいといった要望に応えることが難しい．たとえば，その時間になるたびに，インストールとアンインストールを繰り返したり，アクセス制御の設定を変更する必要がある．もし，管理者がアプリケーションの実行が制限される時間を指定してアクセス制御を行えれば，企業において休み時間中は息抜きのためのアプリケーションの実行を許可するなどの柔軟な運用が可能になる．

Windows や Mac OS には標準でペアレンタルコントロールソフトウェアが搭載されている[3)4)]．ペアレンタルコントロールソフトウェアとは，指定したユーザが実行できるアプリケーションや閲覧できる Web ページを制限するソフトウェアである．ペアレンタルコントロールソフトウェアにより，保護者は子供のコンピュータ利用に対して制限をかけることができ，企業の管理者は社員や職員を統制することができる．ペアレンタルコントロールソフトウェアの中には，実行制限の時間を指定できるものがある．しかし，既存のペアレンタルコントロールソフトウェアには，実行ファイルのコピーなどの方法によって回避されやすいという欠点があった．

そこで本研究では，時間を指定してアプリケーションの実行を制限するシステムを提案する．本システムに対して，実行を制限したいアプリケーションと適用時間，制限の種類に関するポリシーを与える．本システムはそれに従って，アプリケーションの実行を制限する．具体的には，「9 時から 17 時の間にゲームアプリケーションの実行を禁止する」，「12 時から

†1 電気通信大学
University of Electro-Communications

Vol.2011-DPS-146 No.43
Vol.2011-CSEC-52 No.43
2011/3/11

13 時の間にブラウザを実行すると警告を表示する」といったポリシーを管理者が定めると，本システムはそれに従ってアプリケーションの実行を中断したり，警告メッセージを表示したりする．本システムでは，実行ファイルのハッシュ値や，アプリケーションが発行するシステムコールの情報を用いることにより，既存システムに存在していた欠点を克服している．

本論文の構成は次のとおりになっている．2 章では，本システムの設計について述べ，3 章でその実装について述べる．4 章では本システム導入時の評価を報告する．5 章では関連研究を述べ，6 章では本研究のまとめと今後の課題について述べる．

2. 設　　計

本システムは，Linux 上で動作する．本システムは，root 権限を持つ管理者が，root 権限を持たない一般ユーザに対して，ポリシーを適用するために利用されることを想定している．一般ユーザはグラフィカルモードで作業するとする．また，ある程度コンピュータに関する知識を持ち，本システムを回避しようとするユーザは対象外であるとする．

本システムは，一般ユーザがアプリケーションを実行するときに，管理者のポリシーに従ってポリシーのチェックを行う．制限が適用される場合は，管理者のポリシーにしたがって次のどちらかの動作が行われる．

- アプリケーションの実行を禁止する
- アプリケーションの実行時に警告を表示する

実行されなかったことや警告メッセージを伝えるために，**図 1** のようなダイアログを表示する．

本システムでは，ユーザが実行しようとしているアプリケーションの実行ファイルが，ポリシーに記述された実行ファイルと同一であるかを調べるために，実行ファイルのハッシュ値と長さを利用する．ハッシュ値を用いるのは，実行ファイルパスでアプリケーションの識別を行うと，実行ファイルパスが変更された場合に，識別できないからである．また，実行ファイルの長さも合わせて用いることで，実行ファイルの末尾にデータが付け加えられた場合でも，元の実行ファイルのハッシュ値を求めることができる．ハッシュ値の計算には，MD5 を利用する．

アプリケーションのバージョンアップなどにより，実行ファイルの内容が変更された場合や実行ファイルの中の動作に関係しない 1 バイトを変更するなどの回避策に対応するために，本システムは動作中のアプリケーションの挙動を監視し，動的に制限を適用する．アプリケーションの実行開始時から発行されるシステムコールを記録し，ポリシーに記述された

図 1　メッセージ表示ダイアログ．左が実行禁止のときに表示されるダイアログ，右が警告表示のときに表示されるダイアログである．
Fig. 1　The Message dialogs. The left is shown when the system prevents the execution of an application, and the right is shown when the system issues a warning.

システムコールの記録と類似する場合には，そのアプリケーションが実行されているとみなし，制限をかける．ただし，動的な制限の適用はオプションとする．

管理者は，ポリシーファイル作成ツールを用いて，実行を制限したいアプリケーションの実行ファイルを指定する．さらに，その制限の種類に「禁止」または「警告」を指定する．加えて，制限を適用する時間を指定する．ポリシーの例を以下に示す．

```
/usr/bin/sol 70e0841eb0f657b4a250a1184d6de9df 122304 w 9-17
```

この例では順番に，実行ファイルパス，ハッシュ値，サイズ，制限の種類，適用時間が記述されている．制限の種類の `w` は警告 (warning) を表す．適用時間は 9 時から 17 時の間である．このポリシーは，ポリシーファイル作成ツールを利用して作成する．ポリシーファイル作成ツールが，実行ファイルのハッシュ値を求めるため，ユーザがハッシュ値を計算する必要はない．ポリシーに記述されている実行ファイルパスは，ポリシーファイル作成ツールが利用するものである．さらにポリシーファイル作成ツールは，ポリシーファイルに記述されたアプリケーションをある程度の段階まで実行し，アプリケーションが実行の最初で発行するシステムコールの種類と発行回数を規定数分だけ記録したファイルを作成する．

3. 実　　装

3.1 概　　要

本システムは，カーネルモジュールとデーモンからなる．本システムの構成を**図 2** に示す．Linux では，プログラムの実行時に `execve` システムコールが呼び出される．そこで本システムは，`execve` システムコール呼び出しをフックし，処理の流れを変えることで，アプリケーションの実行を制御する．システムコールのフックは，カーネルモジュールにより実現している．

ポリシーのチェックを行うために，ユーザプロセスとして動作するデーモンを立てる．デー

Vol.2011-DPS-146 No.43
Vol.2011-CSEC-52 No.43
2011/3/11

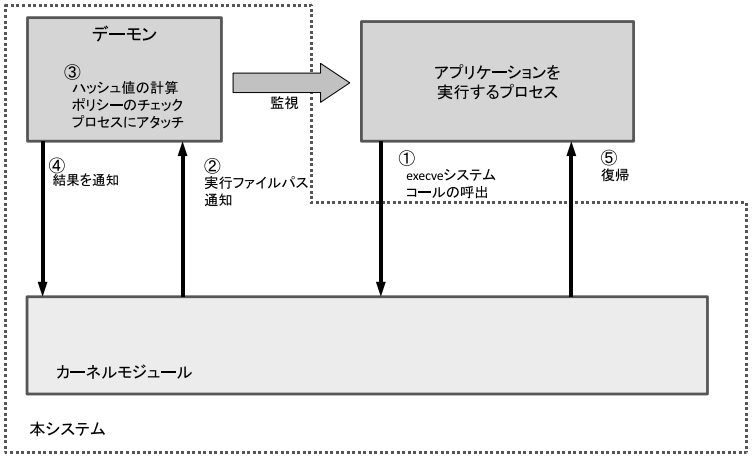

図 2 本システムの構成
Fig. 2 The architecture of the proposed system

モンは，制御端末を持たないプログラムであり，本システムではポリシーのチェックを行う．さらに e オプションを指定して起動すると，アプリケーションのシステムコールに関する挙動を監視して，ポリシーに指定されたアプリケーションが実行されているとみなすと制限を行う．カーネルモジュールとデーモンの通信は，スペシャルファイルを介して行う．デーモンが，スペシャルファイルからデータを読むことで，カーネルモジュールからデータを受け取り，スペシャルファイルに書き込むことでカーネルモジュールへデータを送る．デーモンは，カーネルモジュールから実行ファイルパスを受け取る．デーモンは，ポリシーにしたがって実行ファイルのハッシュ値を計算し，ポリシーのチェックを行う．そして，結果をカーネルモジュールへ通知する．カーネルモジュールは，結果が実行禁止であれば，execve システムコールの返り値として，実行ファイルに実行許可が与えられていないことを表す EACCES を返す．

e オプションが指定され，かつチェックの結果が実行禁止でない場合には，execve システムコールの通常の処理を実行する．デーモンは，アプリケーションの実行の最初の部分のみにおいて ptrace システムコールを利用して，そのアプリケーションのプロセスが発行するシステムコールを記録する．そのデータが，ポリシーファイル作成ツールによって記録された結果に類似していれば，警告メッセージを表示する．

3.2 カーネルモジュール

本システムのカーネルモジュールは，execve システムコールのフックを行うために，システムコールテーブルを書き換える．システムコールテーブルとは，Linux においてシステムコールの呼び出し時に，カーネル関数を呼び出すために参照されるテーブルであり，関数ポインタの配列として実装されている．カーネルモジュールは，システムコールテーブルの execve システムコールに対応する要素の関数ポインタを，本システムが用意した関数のポインタに置き換えることによって，execve システムコールのフックを実現する．システムコールテーブルは，カーネルソースコード中で sys_call_table として定義されているが，2.6 系カーネルでは，セキュリティ上の観点からエクスポートされていない．したがって，カーネルモジュールから sys_call_table を利用することはできない．本システムのカーネルモジュールは，sys_call_table のアドレス付近に存在する適当なシンボルから，システムコールのカーネル関数ポインタの配列を探索することによって，システムコールテーブルを見つけアクセスする．また，システムコールテーブルが置かれているページの属性は書き込み禁止となっている．本システムは，書き換えができるようにページの属性を変更してから関数ポインタの上書きする．

フック関数は，アプリケーションの実行ファイルパスをデーモンに渡し，デーモンからポリシーのチェックの結果を受け取り，その結果に応じてメッセージの表示やアプリケーションの実行を行う．元の Linux において execve システムコールに対応するカーネル関数は sys_execve 関数であり，引数の型は struct pt_regs である．フック関数の引数の型も struct pt_regs regs とする．struct pt_regs は，レジスタを表す構造体であり，execve システムコール呼び出し時は，regs.ebx に実行ファイルパスへのポインタが入っている．この実行ファイルパスは，絶対パスもしくは相対パスである．デーモンが実行ファイルを参照するためには，絶対パスが必要となる．そこでカーネルモジュールは，regs.ebx が示す実行ファイルパスが相対パスであった場合は，絶対パスに変換する．

相対パスから絶対パスへに変換するために，カーネルモジュールは execve システムコールを呼び出したプロセスのカレントディレクトリを求める．プロセスのカレントディレクトリの情報は，プロセス構造体から取得できる．カーネルモジュールは，カレントディレクトリのパスを実行ファイルの相対パスと結合することで絶対パスを求める．

カーネルモジュールとデーモンが通信するためのスペシャルファイルに対しては，デーモンは通常のファイルのように open，read，write，close を行う．その処理の実装はカーネルモジュールに記述される．カーネルモジュールはキャラクタ型デバイスとして動作す

Vol.2011-DPS-146 No.43
Vol.2011-CSEC-52 No.43
2011/3/11

る．スペシャルファイルの open，close，read，write に対応する関数として，cdev_open，cdev_release，cdev_read，cdev_write を実装した．cdev_open では，execve システムコールをフックするためのシステムコールテーブルの書き換えを行い，cdev_release ではシステムコールテーブルを元に戻す．また，cdev_read ではデーモンに実行ファイルパスを渡し，cdev_write ではポリシーチェックの結果を受けとる．

アプリケーションを実行しようとしたプロセスとデーモンのコンテキストは異なる．ゆえに，フック関数とデーモンは互いの間で通信を行う場合に同期をとる．デーモンがスペシャルファイルから読み出しを行う場合は，フック関数が呼び出されて実行ファイルパスが取得されるまで待機する．さらに，フック関数内ではデーモンがポリシーのチェックを終えて結果を返すまで待機する．

本システムでは，待機を行うために Linux カーネルに存在する wait_queue を利用する．wait_queue とは，プロセス構造体を登録すると，プロセスを待機状態にするキューである．本システムでは，デーモンのプロセスが待機を行うときとアプリケーションを実行しようとしたプロセスが待機を行うときに wait_queue を利用する．

デーモンとアプリケーションのプロセスの待機及び復帰の処理を**図 3** に示す．本システムでは，デーモンが待機を行う場合,「execve システムコールが呼び出され，実行ファイルパスをデーモンに渡すことができる状態であること」を条件に指定して，wait_queue を利用して待機を行う．また，アプリケーションを実行しようとしたプロセスが待機を行う場合は，「デーモンの結果通知が完了していること」を条件に指定して待機を行う．本システムでは，デーモンを登録する wait_queue と，アプリケーションのプロセスを登録する wait_queue を利用する．

3.3 デーモン

デーモンの生成には，daemon 関数を利用する．daemon 関数は，自身のプロセスをデーモンにする関数であり，デーモンを作成する処理を内部で行う．daemon 関数の終了後，デーモンはポリシーファイルを読み込み，スペシャルファイルをオープンし，ループ処理に突入する．ループ処理では，スペシャルファイルから実行ファイルパスの読み込みを行う．実行ファイルパスを取得すると，実行ファイルのハッシュ値を計算し，ポリシーのチェックを行う．ポリシーのチェックでは，ハッシュ値の比較および適用時間と現在時刻の比較を行う．チェックが終わると，結果とプロセス ID をスペシャルファイルに書く．プロセス ID は，カーネルモジュールがプロセス構造体を求めるために利用される．

アプリケーションの実行がポリシーに違反する場合は，制限が適用されたことを表すメッセージを表示する．e オプションが指定されており，かつアプリケーションの実行がポリシーに違反しない場合，デーモンはそのアプリケーションの追跡をしながら実行する．デーモンは ptrace システムコールを利用して，アプリケーションプロセスにアタッチし，そのプロセスが発行したシステムコールを監視する．デーモンは，アプリケーションの実行開始時から 1,000 回までのシステムコール呼び出しを記録する．そして，ポリシーファイル作成ツールが作成したファイルから，ポリシーに記述されたアプリケーションのシステムコール発行回数のデータを読み込む．同種類のシステムコール発行回数の差を合計して，本システムが定めた閾値以下である場合に，そのアプリケーションが実行されているとみなす．そうでない場合は，アプリケーションに対してデタッチする．

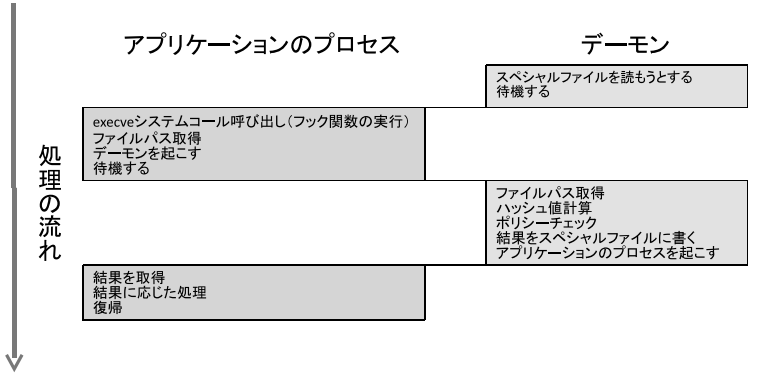

図 3 カーネルモジュールとデーモンの処理の流れ．アプリケーションのプロセスは，execve システムコール呼び出しによって，フック関数を実行し，デーモンと情報のやりとりを行う．
Fig. 3 The execution flow of the kernel module and the daemon. The application process executes the interception function with the execve system call and communicates with the daemon.

4. 評　価

本システムの動作確認とオーバーヘッドの測定，ポリシーに記述されたアプリケーションと類似した動作をするアプリケーションの実行制限の調査を行った．実験環境は，OS が CentOS (カーネル 2.6.18)，計算機の仕様は CPU が Intel Core i7 3GHz，メモリ容量が 8GB である．ポリシーには，6 種類の GNOME ゲームを制限するように指定した．実験で

Vol.2011-DPS-146 No.43
Vol.2011-CSEC-52 No.43
2011/3/11

用いたポリシーを以下に示す.

```
/usr/bin/sol ce55f6d387d42a789039e8d69e6347e1 122304 w 8-20
/usr/bin/gataxx cd08c241dadaa8b59defc895044848a1 63856 w 8-22
/usr/bin/mahjongg 7c6d3a045520a065e0afdb38f18e8831 103556 w 8-22
/usr/bin/gnotski 3ff273c416f3a643de27d588e930f4a1 55896 p 8-22
/usr/bin/glines 3817ec57f271d246240d9b3a18bbc4b1 95844 p 8-20
/usr/bin/gnomine d6da70b29c87af6f404bd74f64357f32 95604 p 8-22
```

4.1 動作確認

ポリシーに記述されたアプリケーションを実行したところ，適用時間内は指定したアプリケーションの実行がポリシーにしたがって制限され，適用時間外は通常どおり実行された．また，ポリシーに記述された実行ファイルをホームディレクトリにコピーし，さらに実行ファイルの末尾にデータを付け加えてそれを実行した．その結果，実行ファイルパスが変更され，実行ファイルの末尾にデータが付加された場合でも実行が制限されることを確認した．

4.2 オーバーヘッド

`execl` のスループットとカーネルコンパイルの実行時間を本システムの導入前と導入後で測定した．導入後に関しては，`e` オプションをつけずにデーモンを起動した場合とつけて起動した場合両方について測定した．`execl` スループットの測定には UnixBench 4.1 を用いた．測定の結果を**表 1** と**表 2** に示す．

測定の結果，本システムを導入した場合，導入前と比較して `e` オプションなしの場合で約 6%，`e` オプションありの場合で約 37% `execl` のスループットが低下した．`execve` システムコール呼び出し時に，デーモンのハッシュ値計算やカーネルモジュールとデーモンの通信処理が加わるため，`execl` のスループットは低下する．`e` オプションありの場合，デーモンが `ptrace` システムコールによって，プロセスの実行中断と再開を行うため，スループットが低下したと考えられる．また，カーネルコンパイルは，`e` オプションなしで 47%増加，`e` オプションありの場合で 76%の増加となった．カーネルコンパイルは，`execve` システムコールを多く呼び出すので，他の多くのアプリケーションではこれ以下のオーバヘッドに収まると考えられる．`execve` システムコールが短時間に大量に呼び出されるアプリケーションは少ないため，本システムを導入した場合に OS 利用の体感速度が大幅に低下する現象は実験では見られなかった．

表 1 `execl` のスループット
Table 1 Execl throughput

	導入前	e オプションなし	e オプションあり
execl のスループット (loops/s)	2082.1	1962.8	1313.6

表 2 カーネルコンパイルの実行時間
Table 2 Kernel compile time

	導入前	e オプションなし	e オプションあり
カーネルコンパイルの時間 (s)	1090	1605	1913

表 3 実行制限が適用された率
Table 3 Rate of execution restriction

	sol	gataxx	mahjongg	gnotski	glines	gnomine
制限適用率 (%)	87	73	70	67	67	73

性能低下に対する改善策としては，一般ユーザが書き換えられないファイルに関しては，一度計算した実行ファイルのハッシュ値を保存しておき，2 回目以降はその値を利用するという方法が考えられる．

4.3 システムコール列によるアプリケーションの認識

`e` オプションを指定した場合の動作について実験した．ポリシーに記述してあるハッシュ値の一部を書き換えることによって，実行ファイルのハッシュ値が，ポリシーと異なる場合を擬似的に再現して，ポリシーに記述された実行ファイルを実行した．実験では，実行ファイルを 30 回実行し制限が適用された回数を数え，それを元に制限適用率を求めた．3.3 節で述べた閾値は 100 とした．実験の結果を**表 3** に示す．制限適用率は，67%から 87%という結果となった．実行が制限されない場合が多く見られた原因については現在調査中である．

さらに，false positive がどの程度観測されるか調べるために，ポリシーに記述されていない 35 個のアプリケーションを実行した．アプリケーションは，GNOME のアプリケーションメニューに登録されているものである．その結果，35 個中 14 個のアプリケーションで，ポリシーに記述がないにもかかわらず制限が適用された．こちらについても現在原因を調査中である．

実験では，システムコール列を用いたアプリケーションの認識について高い精度を得ることはできなかった．現在無視できない割合の false negative，false positive が観測されており，今後改善が必要である．

Vol.2011-DPS-146 No.43
Vol.2011-CSEC-52 No.43
2011/3/11

5. 関 連 研 究

TIFPS[5] は，既存のアクセス制御に対して，時間属性を追加したアクセス制御システムである．既存のアクセス制御では，ある特定の時間だけファイルへのアクセスを許可することはできないが，TIFPS を用いればアクセス制御に時間的な制約を用いることができる．TIFPS では，サブジェクト（プロセス）とオブジェクト（ファイル）に時間属性が与えられる．TIFPS の下でサブジェクトがオブジェクトにアクセスする場合，オブジェクトの時間属性は，サブジェクトの時間属性とオブジェクトの時間属性の積集合をとったものとなる．これが原因で，オブジェクトにアクセスできる時間が短くなってゆくという問題がある．また，ファイルコピーの方法によっては，時間属性が新しいファイルに正しく継承されないという実装上の問題がある．本研究は，ファイルやディレクトリに対する汎用のアクセス制御ではなく，プログラムの実行の制限に特化したアクセス制御を提案している．また，制御はファイル等に属性を付加するのではなく，実行ファイルのハッシュ値や長さを用いることによって行っている．

TOMOYO Linux[1] や SELinux[2] はセキュリティを高めた Linux であり，ポリシーにしたがって強制アクセス制御を行う機能を提供する．これらの OS はセキュア OS と呼ばれることがある．セキュア OS を利用する際には，正しいポリシーの管理運用が必要である．したがって，管理者にはポリシーの管理運用という負担が課せられる．TOMOYO Linux では，この負担を軽減するために，ポリシーの自動学習を行う機能を備えている．また，ポリシーの設定が直感的にわかりやすいように実装されている．本研究のポリシーは，アプリケーションの実行制限に特化しているため，セキュア OS のポリシーよりも記述や理解が容易である．また，本研究のシステムでは，既存のセキュア OS とは異なり，時間に応じて実行制限の有無を変えることができる．

本研究のシステムコール列を用いたアプリケーションの認識は，システムコールの履歴に基づく異常検知と極めて近い分野である．異常検知の分野では，システムコールの履歴に基づく異常検知について，N-gram による手法[6] やシステムコールの引数や返り値を求める手法[7] などが提案されている．このような方法を応用すれば，アプリケーションの動的な識別について高い精度が得られる可能性がある．

6. お わ り に

本研究では，ポリシーを設定することで，アプリケーションの実行を制限するシステムを提案した．本システムによって，指定したアプリケーションの実行を指定時間のみに制限することができることを確認した．本システムでは，アプリケーションの識別に，実行ファイルのハッシュ値を用いている．したがって実行ファイルの内容が，アプリケーションのバージョンアップなどによって変更された場合，ハッシュ値も変わるため，ハッシュ値のみによるアプリケーションの識別は効果が限定される．そこで，本システムはシステムコールの発行回数を利用することで，指定したアプリケーションと類似した動作をするアプリケーションの利用に対して，警告の表示をした．しかしながら，実験では false negative，false positive が共に多く，今後精度の向上が必要であることがわかった．

今後の課題を以下に示す．本システムでは，アプリケーションの実行制限の適用時間外にアプリケーションを起動し，適用時間に入ってもそのままを使いつづけた場合に対して，特に何もしていないので，警告を表示するなどの動作を加えたい．また，ポリシーの項目に関して，ユーザを指定できたり，時間を分や秒単位で指定できるなど，より柔軟なポリシーを運用できるように拡張したいと考えている．

参 考 文 献

1) 原田　季栄，半田　哲夫，板倉　征男：TOMOYO Linux の設計と実装，第 21 回コンピュータシステム・シンポジウム (ComSys2009) 論文集，pp.101–110 (2009).
2) Smalley, S., Vance, C. and Salamon, W.: Implementing SELinux as a Linux Security Module, *NAI Labs Report*, pp.1–43 (2006).
3) 保護者による制限 - Windows 7 の機能，http://windows.microsoft.com/ja-JP/windows7/products/features/parental-controls.
4) Mac OS X:「許可される操作」、「制限」、「ペアレンタルコントロール」を使ってアプリケーション、システム環境設定、ディスク作成へのユーザアクセスを管理する方法，http://support.apple.com/kb/HT2400?viewlocale=ja_JP.
5) Chiang, K., Nguyen, T.D. and Irvine, C.E.: A Linux Implementation of Temporal Access Controls, *In Proceedings of the 2007 IEEE Workshop on Information Assurance*, pp.309–316 (2007).
6) Forrest, S., Hofmeyr, S.A., Somayaji, A. and Longstaff, T.: A Sense of Self for Unix Processes, *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pp.120–128 (1996).
7) Kruegel, C., Mutz, D., Valeur, F. and Vigna, G.: On the Detection of Anomalous System Call Arguments, *In Proceedings of ESORICS 2003*, pp.326–343 (2003).