LSM のオーバーヘッド測定によるセキュア OS の性能評価

松田　直人 †　　田端　利宏 †　　宗藤　誠治 ‡

† 岡山大学 大学院自然科学研究科
700-8530 岡山県岡山市津島中 3–1–1
matsuda@swlab.it.okayama-u.ac.jp
tabata@cs.okayama-u.ac.jp

‡ 日本アイ・ビー・エム東京基礎研究所
242-8502 神奈川県大和市下鶴間 1623–14
munetoh@jp.ibm.com

あらまし システムの脆弱性の悪用により，企業システムや個人用計算機が様々な被害を受ける問題が起こっている．この問題を解決する手段として，セキュアを用いる方法が有効である．しかし，その実現方式と性能について語られることは少ない．このセキュア OS の機能は，Linux 2.6 以降では，Linux Security Modules（LSM）により実装される場合が多い．そこで，我々は，LSM のオーバーヘッド測定のための機能拡張を Linux カーネルに行い，代表的な 4 つのセキュア OS の性能評価を行った．本論文では，ベンチマークソフトによる各セキュア OS のオーバーヘッドと，各 LSM フックでのオーバーヘッドの分析結果について述べる．

An Evaluation of Performance of Security Focused OS by Measuring the Overhead of LSM

Naoto Matsuda†　　Toshihiro Tabata†　　Seiji Munetoh‡

†Graduate School of Natural Science and Technology, Okayama University
matsuda@swlab.it.okayama-u.ac.jp
tabata@cs.okayama-u.ac.jp
‡IBM Tokyo Research Laboratory
munetoh@jp.ibm.com

Abstract Enterprise systems and individuals suffer from various attacks because of exploiting vulnerabilities of the systems. As a method resolving this problem, security focused operating system (OS) is an effective way. However, the performance of security focused OS has little chance to talk about. In Linux 2.6, since the function of security focused OS is implemented using Linux Security Modules (LSM), we have extended the function to measure the overhead of the LSM, and evaluated the performance of four typical security focused OS. In this paper, we describe the result of analysis of the overhead of security focused OS and the LSM hooks.

1　はじめに

システムの脆弱性の悪用により，企業システムや個人用計算機が様々な被害を受ける問題が起こっている．特に，root 権限が奪取されてしまうと，その被害は甚大なものとなってしまう．しかしながら，ファイアウォールや IDS などを用いた従来の方法で，これらの攻撃をて防ぐのは非常に難しい．また，ゼロデイ攻撃への対処も難しい．

これらの問題を解決する手段として，セキュアオペレーティングシステム（OS）を用いる方法が有効である．セキュア OS は，強制アク

セス制御（MAC）機能を提供することで，たとえ root 権限が奪取されたとしても，被害をポリシで許可された範囲内にとどめることができる．Linux[1]におけるセキュア OS は，Linux 2.6 にて追加された機能である Linux Security Modules（LSM）[1] により，実装される場合が多い．その実装例として，Security-Enhanced Linux（SELinux）[2] や AppArmor[3] などがある．

これらのセキュア OS 間には，実装方式や資源の識別方式の違いによるセキュリティ特性について，比較がなされている．しかしながら，その性能について語られることは少ない．システムとそれに対する脅威は高度化・複雑化しているため，新たな脅威への対抗策としても，セキュア OS は有効である．しかし，実際のシステムでセキュア OS を使用する場合には，その性能を把握することが必要不可欠である．

そこで，我々は，LSM のオーバーヘッド測定機能である LSM Performance Monitor（LSMPMON）を Linux カーネルに実装した．LSMPMON は，各 LSM のフック箇所での処理時間と呼び出し回数を記録することで，セキュア OS 間での性能比較を容易に行うことが可能である．本稿では，この LSMPMON の実現方式と，本機能を用いたセキュア OS の評価結果について述べる．

2 技術背景

2.1 セキュア OS

2.1.1 概要

セキュア OS は，一般的に，強制アクセス制御（MAC）と最小特権を実現する機能を備えた OS のことを指す．多くの OS で用いられている任意アクセス制御（DAC）では，各ファイルの所有者が独自にアクセスパーミッションを設定できる．また，UNIX における root や Windows における Administrator のようなスーパーユーザ権限を持つユーザは，DAC のチェックをバイパスして全ての資源にアクセス可能である．このため，悪意のあるユーザによってスーパーユーザ権限が奪取されてしまうと，その被害は甚大なものとなってしまう．

[1]Linux は、Linus Torvalds の米国およびその他の国における商標.

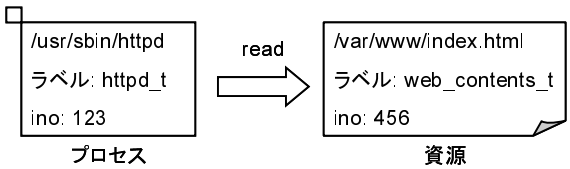

図 1 Web サーバによる Web ページの読み込み

一方，MAC を用いると，セキュリティポリシによって許可された資源への許可された操作のみに行動が限定される．これは，スーパーユーザ権限を持つユーザにも強制されるため，不正アクセス等によりスーパユーザ権限を得たとしても，その行動を制限できる．

セキュア OS のうち，対象 OS が Linux であり，かつ，オープンソースで開発されている代表的なものとして，SELinux, AppArmor, TOMOYO Linux[4]，及び LIDS[5] がある．これらのセキュア OS の特徴と資源の管理方法の違いを，Web サーバ（`httpd`）による Web コンテンツの読み込み（図 1）を例に挙げて以降で説明する．

2.1.2 SELinux

SELinux は，米国家安全保障局によって開発されているセキュア OS であり，Linux カーネルに標準で組み込まれている唯一のセキュア OS である．SELinux は，資源の識別方式にラベルを用いており，図 1 の例では，httpd_t というラベルを付与されたプロセスが，web_contents_t ラベルを付与されたファイルの読み込みを行うと解釈される．

2.1.3 AppArmor

AppArmor は，米 Novell 社を中心に開発されているセキュア OS である．資源の識別には，パス名を用いており，`/usr/sbin/httpd` が，`/var/www/index.html` の読み込みを行うと解釈される．

2.1.4 TOMOYO Linux

TOMOYO Linux は，NTT データによって開発されているセキュア OS である．資源の識別には，AppArmor と同様にパス名を用いている．また，プロセスの識別には，パス名とその

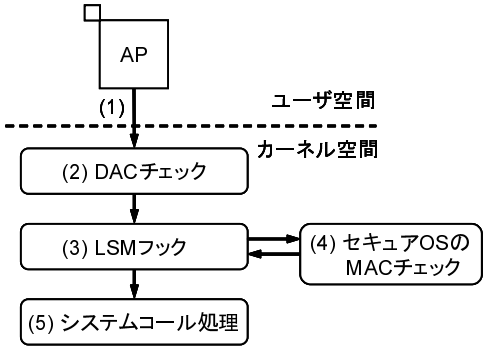

図 2 LSM の構造

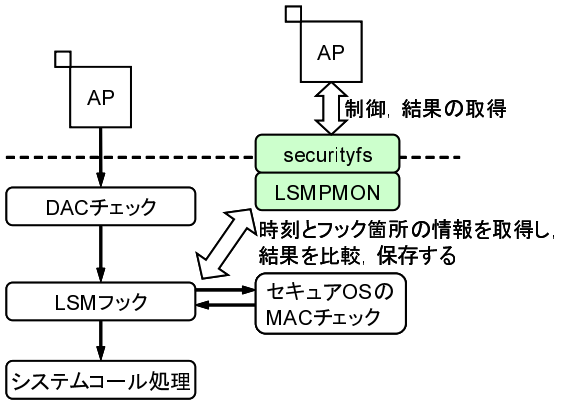

図 3 LSMPMON の構造

プロセスがどういった経路で実行されたかという履歴を使用している．例えば，同じ `httpd` であっても，OS 起動時にカーネルによって起動された `httpd` と，ユーザがシェルから起動した `httpd` に対して，異なるポリシを付与することができる．

2.1.5 LIDS

LIDS は，Xie Huangang 氏と Philippe Biondi 氏によって開発されたセキュア OS である．LIDS は，アクセス制御の設定にはパス名を用いている．一方，内部的には，i ノード番号を用いて資源の管理を行っている．しかし，i ノード番号による管理には問題がある．vi や emacs などの多くのテキストエディタは，ファイルの編集のために一時ファイルを作成し，データの変更はその一時ファイルに対して行われる．そして，変更を保存する際には，一時ファイルで元のファイルを上書きする．これにより，ファイル名は同じであるが，i ノード番号の違うファイルができてしまうため，正しいアクセス制御を行うことができなくなってしまう．

2.2 LSM

LSM は，カーネル内のセキュリティチェック機構へのフック関数群を定義する機能である．Linux 2.6 以降では，LSM がカーネルに組み込まれており，セキュア OS の機能は，LSM により実装される場合が多い．LSM の構造を図 2 に示し，以下に説明する．アプリケーションプログラム（AP）がシステムコールを発行すると，最初に，DAC によるセキュリティチェックが行われる．次に，カーネル内のセキュリティチェック箇所（Linux 2.6.19 現在，163 箇所）で，登録された LSM のフック関数が呼び出され，各セキュア OS によりセキュリティチェックが行われる．これらのチェックで操作が許可された場合のみ，実際のシステムコール処理が行われる．

3 LSMPMON

3.1 目的

セキュア OS を実際のシステムで使用する場合，その性能を把握することが必要不可欠であり，システムに適したセキュア OS の選択には細かな性能比較が必要である．セキュア OS のセキュリティチェック機能は，LSM フック関数中で呼び出される．よって，セキュア OS の性能評価を行うには，LSM フック関数の性能評価を行えばよいことがわかる．そこで，LSM のオーバーヘッド測定機能である LSM Performance Monitor（LSMPMON）を Linux カーネルに実装した．

3.2 実現する機能

性能測定のために最低限必要な要件は，フック関数の処理時間と呼び出し回数の測定を漏れなく行えることである．また，本機能に求められる要望として，動作の制御と結果の取得が容易であり，測定機能のオーバーヘッドが小さいことが挙げられる．

3.3 実装

我々は，LSMPMON を Linux 2.6.19.7 に実装した．この LSMPMON の構造を図 3 に示す．LSMPMON は，全 LSM フック関数の呼び出し

表 1 各セキュア OS の比較

	SELinux	AppArmor	TOMOYO	LIDS
LSM フック数	149	39	17	38
資源の識別方式	ラベル	パス名	パス名 + 実行履歴	i ノード
バージョン	2.4.6-80.fc6	March 07 v405	2.0	2.2.3rc1

表 2 LSMPMON のオーバーヘッド（単位:sec）

	有効時	無効時
ApacheBench の実行時間	0.357	0.348

前後での時刻を取得し，対象の LSM フック関数の処理時間を測定する．セキュア OS の全てのセキュリティチェックは，LSM フック関数を通して行われる．よって，漏れなく測定を行えるため，機能要件を満たしているといえる，

機能要望であるユーザインタフェースの簡易化には，securityfs を用いることによって対応した．securityfs は，セキュリティモジュール用の特殊な仮想ファイルシステムであり，ユーザ空間とカーネル空間でのデータのやり取りを簡易に行うことができる．

LSMPMON の動作例を図 4 に示す．securityfs を用いることで，ユーザ空間とカーネル空間でのデータの授受を `echo` や `cat` のような既存の AP を用いて行えることがわかる．表示する情報は，各 LSM フック関数の最短処理時間（min）と最長処理時間（max），平均処理時間（ave）及び呼び出し回数（count）である．これらの情報を用いることにより，セキュア OS 間の性能比較だけでなく，ボトルネック箇所の把握も可能であるため，開発時にも有用である．

LSMPMON の基本オーバーヘッドを表 2 に示す．性能測定には，ApacheBench を用い，ファイルの読み込み要求を 100 回行ったときの実行時間を示している．この結果より，LSMPMON のオーバーヘッドは，約 2.5%と小さい．

以上のことより，簡易なユーザインタフェースの実現と，機能の軽量化という機能要望を満足しているといえる．

```
  LSMPMON 有効化
% echo 1 > /sys/kernel/security/lsmpmon/control

  LSMPMON 無効化
% echo 0 > /sys/kernel/security/lsmpmon/control

  結果を表示
% cat /sys/kernel/security/lsmpmon/result
hook              min     max       ave     count
-----------------------------------------------

      ⋮
inode_create      97     1818075   105   3035605
inode_link
inode_unlink      97     80903     114   3035600
      ⋮
```

図 4 LSMPMON の動作例

4 評価

4.1 概要

我々は，LSMPMON を用いて，SELinux, AppArmor, LIDS, 及び TOMOYO Linux の性能評価を行った．性能評価には LMbench[6] を用い，ファイル操作におけるセキュア OS のオーバーヘッドと，主要なファイル操作時にセキュリティチェックを行う LSM フック関数のオーバーヘッドについて評価を行った．

4.2 評価内容

評価は，CPU: Intel Pentium 4 (3.0GHz)，メモリ: 1GB, OS: Linux 2.6.19.7 の計算機上で行った．評価対象として，セキュア OS を使用しない場合（None）と 2 章で挙げた 4 つのセキュア OS を用い，以下の 2 点について評価を行った．

評価（1） LMbench によるオーバーヘッド比較
ベンチマークソフト LMbench を使用し，その中でも特にファイル操作に関する結果を評価する．測定は 5 回行い，その平均処理時間を示す．

評価（2）LSMPMON による LSM フック関数のオーバヘッド比較
ファイル操作時にセキュリティチェックを行う LSM フック関数のオーバーヘッドを LSMPMON を用いて評価する．測定には LMbench を 5 回実行し，その平均処理時間を示す．

なお，各セキュア OS の資源識別方式と LSM フック箇所数，セキュア OS のバージョンは，表 1 のとおりである．

4.3 測定結果

評価（1）の結果を表 3 に示す．

SELinux における stat の処理時間は，約 58% 増加している．ファイル処理のうち，最も多用されるものは stat であり，その割合は約 40%であるという調査結果もある [7]．このため，システム全体のオーバーヘッドも大きくなる可能性が高い．

ファイルの生成時間に着目すると，SELinux の処理時間増加率が最も高い．このため，大量のファイル生成が起こるメールサーバなどの用途では注意が必要である．しかし，より多くのディスクアクセスが必要になる 10KB のファイル生成では，処理時間増加率は低下するため，サイズの大きいファイルを扱う場合には，その影響は小さいと考えられる．

評価（2）の結果を表 4 に示す．なお，表 4 中の “N/A” は，対象のフック関数が実装されていないことを表す．

処理時間が 1,000 クロックサイクル未満ものは，主にポリシの探索のための時間である．
SELinux は，inode_create の処理時間が非常に長い．これは，新たに作成される i ノードに対して，ラベル付けを行う必要があるためである．AppArmor は，inode_create と inode_unlink 処理に，他の処理の約 10 倍の時間を要している．これは，パス名の取得が必要なためと考えられる．TOMOYO Linux は，inode_permission 処理とその他場合に要する時間では，約 20 倍の差がある．これは，inode_permission 以外の処理時に，セマフォ獲得の必要があるためである．

一方，LIDS は，i ノード番号によって資源の管理を行っているため，ポリシの探索を高速に行うことができる．よって，LSM フック関数の処理時間は，比較的短いものになっている．

次に，表 3 中の各処理で呼び出される LSM フック関数のうち，表 4 に含まれるものの関数名と呼び出し回数を表 5 に示す．ただし，LSM フック関数の処理時間とその呼び出し回数は，システムコールのオプション等によって異なる．

表 3，4，5 より，これらの間には，相関関係があることが分かる．SELinux の 0K file create は，非常に低速である．この理由は，file create の処理中に呼び出される inode_create の処理が低速なためである．TOMOYO Linux は，open/close の処理時間増加率が非常に高い．これは，ファイル open 時の inode_permission の処理時間は，約 1.76μsec と，平均値よりも非常に大きくなっているためである．しかし，TOMOYO Linux は，read や write といったファイル open 後の操作でセキュリティチェックを行っていない．このため，10KB のファイル生成処理時間の増加率は小さくなっている．

4.4 まとめ

ラベルによる識別を行う SELinux は，ラベル付けを行う必要があるため，ファイル生成時のオーバーヘッドが大きく，ファイル生成時以外のオーバーヘッドも，比較的大きい．また，設定工数が多く，設定内容が複雑で理解しにくいという問題もある．しかし，粒度の細かいアクセス制御設定が可能であることや，情報フローの分析を確実に行えるというメリットがある．

パス名による識別を行う AppArmor と TOMOYO Linux は，パス名の取得が必要な場合のオーバーヘッドが大きくなる．また，情報フロー分析が困難である．しかし，パス名を用いることにより，設定が分かりやすく，ポリシの自動生成にも有利である．

i ノードによる識別を行う LIDS のオーバーヘッドは，比較的小さい．しかし，i ノード番号が変化する場合に問題が発生するため，対処が必要である．

5 おわりに

LSM のオーバーヘッド測定機能である LSMPMON の実装と，本機能を用いたセキュア OS の

表 3 ファイル操作時の処理時間と増加率（単位:μsec）

	None	SELinux	AppArmor	TOMOYO	LIDS
stat	1.67	2.65 (58%)	1.87 (12%)	2.02 (21%)	2.17 (30%)
open/close	2.49	3.62 (45%)	4.16 (67%)	8.76 (252%)	3.27 (31%)
0K file create	9.67	25.92 (168%)	14.16 (47%)	16.00 (66%)	13.90 (44%)
0K file delete	5.58	8.61 (54%)	8.03 (44%)	9.88 (77%)	6.68 (20%)
10K file create	36.82	51.86 (41%)	40.30 (9%)	41.50 (13%)	39.38 (7%)
10K file delete	15.82	19.66 (24%)	19.44 (23%)	21.40 (35%)	18.62 (18%)

表 4 ファイル操作時の LSM フック関数の処理時間（単位:μsec）

	None	SELinux	AppArmor	TOMOYO	LIDS
inode_create	0.035	4.411	1.701	3.047	N/A
inode_unlink	0.038	0.270	1.523	2.803	0.096
inode_permission	0.034	0.152	0.110	0.142	0.156
inode_setattr	0.041	0.285	0.261	2.899	0.152
inode_getatttr	0.035	0.124	0.035	N/A	N/A
file_permission	0.037	0.044	0.200	N/A	0.040

表 5 各ファイル操作時に呼び出される LSM フック関数とその呼び出し回数

	LSM フック関数	回数
stat	inode_permission	2
	inode_getattr	1
open/close	inode_permission	3
file create	inode_permission	4
	inode_create	1
	inode_setattr	1
	file_permission	1
file delete	inode_permission	2
	inode_unlink	1

評価について述べた．具体的には，Linux におけるセキュア OS の実装方式について説明し，そのオーバーヘッドを測定する方法について述べた．測定機能を Linux 上へ実装し，4 つのセキュア OS について評価を行った．評価の結果，セキュア OS の実装方式により，性能に大きな差があることを確認した．

今後の課題として，他の LSM フック関数のオーバーヘッド比較と，オーバーヘッドの詳細な分析，及び LSMPMON の機能拡充がある．

謝辞 本研究の一部は，C&C 振興財団 若手研究員助成，及び科学技術振興機構戦略的国際科学技術協力推進事業の支援を受けて行った．

参考文献

[1] C. Wright, C. Cowan, J. Morris, S. Smalley, G. Kroah-Hartman, “Linux Security Modules: General Security Support for the Linux Kernel,” Proceedings of 11th Annual USENIX Security Symposium, pp 17–31, 2002.

[2] P. Loscocco, S. Smalley, “Integrating flexible support for security policies into the Linux operating system,” Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX ’01), pp 29–42, 2001.

[3] Novell, AppArmor, http://www.novell.com/linux/security/apparmor/

[4] 原田 季栄，保理江 高志，田中 一男，“使いこなせて安全な Linux を目指して，” Linux Conference 2005.

[5] LIDS, http://www.lids.org/

[6] LMbench, www.bitmover.com/lmbench/

[7] D. Roselli, J. R. Lorch, T. E. Anderson, “A Comparison of File System Workloads,” Proceedings of 2000 USENIX Annual Technical Conference, pp. 41–54, USA, June, 2000.