情報処理学会論文誌 Vol. 52 No. 9 2596–2601 (Sep. 2011)

テクニカルノート

性能評価機構LSMPMONによるセキュアOSの評価

山本賢治[†1] 山内利宏[†1]

Linuxでは，セキュアOSにはいくつかの実装方式があり，かつ複数の選択肢があるものの，Linuxカーネルのバージョンの違いによるセキュアOSの性能の変化や，Linuxカーネル2.6.19より新しいLinuxカーネルでのセキュアOSの詳細な性能は明らかでない．そこで，本論文では，LSMのオーバヘッド測定用ツールLSM Performance Monitorを Linuxカーネル2.6.30に対応させ，各実装方式の代表的な3つのセキュアOS（SELinux，TOMOYO Linux，LIDS）の比較評価を行った結果を報告する．これにより，Linuxカーネルのバージョンの違いによる性能への影響，およびより新しいLinuxカーネルでのセキュアOSの性能について報告する．

Evaluation of Performance of Secure OS Using Performance Evaluation Mechanism of LSMPMON

Kenji Yamamoto[†1] and Toshihiro Yamauchi[†1]

In some projects, secure OSes for Linux have been developed and different implementations have been adopted. However, there is no report on evaluation of performance of secure OS that after Linux 2.6.19 in detail, and the relationship between the kernel version and the performance is not clear. Therefore, we evaluate change of the performance at the version interval and overhead of three secure OSes (SELinux, TOMOYO Linux, LIDS), by using the overhead measurement tool, the LSM Performance Monitor (LSMPMON) developed for Linux 2.6.30. This paper shows the performance of secure OSes on Linux 2.6.30.

†1 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

1. はじめに

オペレーティングシステム（以降，OSと略す）のセキュリティを強化する方法として，セキュアOSが注目されている．セキュアOSは，強制アクセス制御（Mandatory Access Control，以降，MACと略す）と最小特権を提供することで，たとえスーパユーザ権限が攻撃者に奪取されたとしても，被害を最小限にすることができる．

セキュアOSには，ラベル方式に基づくものやパス名方式に基づくもの等，資源の識別方式の異なる複数の実装方式が存在する．このため，利用者が，どのセキュアOSを用いるのが適切なのかを判断するのは簡単ではない．また，セキュアOS導入による性能への影響は明確でない．たとえば，OSのバージョンの変更にともなう仕様や性能の変化がセキュアOSに与える性能面の影響も明らかでない．さらに，セキュアOSは組み込み用途でも注目されており，性能面での評価は重要である．

そこで，Linux 2.6においては，セキュアOSの機能がLinux Security Modules[1)]（以降，LSMと略す）というフック関数群により実装されることが多いことに着目し，LSM Performance Monitor（以降，LSMPMONと略す）と名付けたLSMで実装されたセキュアOSの性能評価機能を実現した[2)]．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．これにより，セキュアOS間での性能比較が容易に行える．

本論文では，LSMPMONをLinuxカーネル2.6.30に対応させ，セキュアOSの詳細な性能評価を文献2)より新しいLinuxカーネルで行った結果を報告する．評価には，ラベル方式，パス名方式，およびiノード方式の各方式の代表的なセキュアOSを用いた．性能評価では，Linuxカーネル2.6.30で文献2)と同様の評価を行い，文献2)で評価されているLinuxカーネル2.6.19の結果と比較することで，性能面での変化を明らかにする．また，Linuxカーネル2.6.30におけるセキュアOSの性能について報告する．

2. セキュアOS

2.1 評価に用いたセキュアOS

セキュアOSは，MACと最小特権を実現する機能を備えたOSのことを指す．対象OSがLinuxであり，LSMを利用して開発されているセキュアOSを評価対象とした．具体的には，Security-Enhanced Linux[3),4)]（以降，SELinuxと略す），TOMOYO Linux[5),6)]，およびLIDS[7)]を用いた．これらのセキュアOSの特徴と資源の識別方式の違いを**図1**を用いて説明する．図1は，Webサーバによる Webコンテンツの読み込み処理の例である．

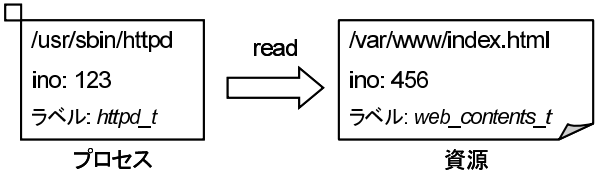

図 1　Web サーバによる Web ページの読み込み
Fig. 1　Process by which the Web server reads a Web page.

セキュア OS の資源の識別方式には，ラベル，パス名，および i ノード番号で管理する方法がある．

SELinux は，資源の識別にラベル方式を用いており，ドメイン，タイプと呼ばれるラベルをそれぞれプロセス，ファイルに割り当てることで制御を行う．SELinux は，TE（Type-Enforcement），MAC，RBAC（Role-Based Access Control）により，安全性を高めている．図 1 の例では，httpd_t というドメインを付与されたプロセスが，web_contents_t タイプを付与されたファイルの読み込みを行うと解釈される．

TOMOYO Linux は，資源の識別方式にパス名方式を用いており，パス名とそのプロセスがどういった経路で実行されたかという履歴によって制御を行う．図 1 の例では，/usr/sbin/httpd が，/var/www/index.html の読み込みを行うと解釈される．

LIDS は，資源の識別方式に i ノード方式を用いている．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が，アクセス制御の設定にはパス名を用いている．

2.2　LSM

LSM は，Linux 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アクセス制御フレームワークである．この機能を用いることにより，カーネルを修正することなくカーネルのセキュリティチェック機能をユーザが独自に拡張することが可能である．LSM を有効にすると，カーネル内部のオブジェクトへアクセスする前に，ユーザが登録した LSM のコールバック関数が呼び出され，安全性のチェックが行われる．

3.　LSMPMON

文献 2) で提案した LSMPMON の主な機能について説明する．

(1) LSM フック関数単位での性能測定の実現

LSMPMON の構造を**図 2** に示す．LSMPMON は，すべての LSM フック関数の呼び出し前後で時刻を取得し，対象の LSM フック関数の処理時間と呼び出し回数を測定する．

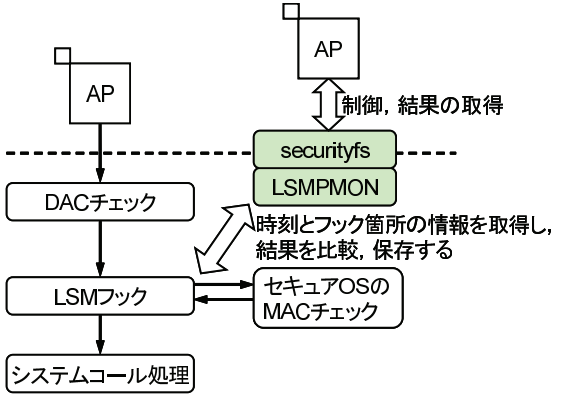

図 2　LSMPMON の構造
Fig. 2　Structure of the LSMPMON.

(2) コンテキストスイッチの検知

LSM フック関数の処理中のコンテキストスイッチを検知するためのフラグを用意した．このフラグの値を確認することでコンテキストスイッチの有無を把握する．

(3) 簡易なインタフェースによる制御

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，securityfs を用いることにより，LSM を使用するモジュールは，独自のファイルシステムを作成することなくユーザ空間とカーネル空間でのデータのやりとりを簡易に行うことができる．

(4) 測定対象の限定機能

(3) であげた securityfs を利用することで，測定対象を限定機能の制御を行う．securityfs 上の特定のファイルに対し，“s foo” という文字列を書き込めば，サブジェクト名が “foo” の場合のみ測定結果を保存する．オブジェクトの場合も同様に，“o bar” という文字列を書き込めば，オブジェクト名が “bar” の場合のみ測定結果を保存する．

(5) システムコール情報の利用機能

LSMPMON は，audit context を参照してシステムコール番号を取得し，システムコールごとに LSM フック関数の処理時間を測定し，保存することができる．

4. LSMPMON を用いたセキュア OS の評価

4.1 評価の観点

新たに Linux カーネル 2.6.30 に対応させた LSMPMON を用いて，Linux カーネルのバージョンの違いやセキュア OS の導入による性能への影響を明らかにするため，以下の 3 つの観点で評価した．

(A) Linux カーネルのバージョンの違いによる性能と仕様の変化
(B) セキュア OS 導入による性能への影響
(C) セキュア OS における資源の識別方式の違いによる特徴

4.2 評価の方法

セキュア OS のアクセス制御処理が頻発するファイル操作について評価した．その内容を以下に述べる．

(A) Linux カーネルのバージョンの違いによる性能と仕様の変化

（評価 1）ベンチマークソフト LMbench[8] を用いて，Linux カーネル 2.6.30 における各セキュア OS の性能を測定し，その中のファイル操作に関する結果を評価した．この評価結果と文献 2) の Linux カーネル 2.6.19 の評価を比較し，各セキュア OS の性能の変化について考察する．

しかし，LMbench における評価だけでは，性能の変化の要因，ボトルネック箇所を把握することができない．そこで，（評価 1）において性能が変化した項目について，以下の評価を行った．

（評価 2） LSMPMON を用いたシステムコールごとの詳細な比較を行った．個別のシステムコール内の LSM フック関数のオーバヘッドを LSMPMON によって測定し，その結果を異なるバージョン間で比較評価した．これにより，詳細なオーバヘッドの比較を行い，Linux カーネルのバージョンの違いによる性能と仕様の変化を示す．

(B) セキュア OS 導入による性能への影響

（評価 1）の結果をもとに，セキュア OS の導入による性能への影響を考察する．

(C) セキュア OS における資源の識別方式の違いによる特徴

（評価 1）の結果だけでは各フック関数の処理時間や，ボトルネック箇所を把握することができない．そこで，以下の評価を行った．

（評価 3）LSMPMON を用いて LSM フックごとの処理時間を測定し，比較評価した．これにより，資源の識別方式の違いによる影響を LSM フック単位で明らかにする．

表 1　新カーネルにおける各セキュア OS の比較
Table 1　Compare to each secure OS in the new kernel.

	SELinux	TOMOYO Linux	LIDS
Use of LSM hooks	154	14	46
Identification method of the resource	label	path-name + execution history	inode
Version	3.6.12-39.fc11	2.2.0	2.2.3rc8

表 2　旧カーネルにおける各セキュア OS の比較
Table 2　Compare to each secure OS in the old kernel.

	SELinux	TOMOYO Linux	LIDS
Use of LSM hooks	149	17	38
Identification method of the resource	label	path-name + execution history	inode
Version	2.4.6-80.fc6	2.0	2.2.3rc1

また，評価環境は，文献 2) の評価と同じ計算機（CPU：Pentium 4（3.0 GHz），メモリ：1 GB）である．以降では，Linux 2.6.30.4 を新カーネル，Linux 2.6.19.7 を旧カーネルと表記する．なお，測定はすべて LMbench を 5 回実行し，その平均処理時間を示す．各セキュア OS のアクセス制御を行う対象と資源の識別方式，LSM フックの利用数，およびセキュア OS のバージョンは，**表 1** と**表 2** のとおりである．

4.3 Linux カーネルのバージョンの違いによる性能と仕様の変化

4.3.1 異なるバージョンのカーネルによる性能比較

（評価 1）の結果を**表 3** に，文献 2) の評価結果に read/write の測定結果を加えたものを**表 4** に示す．SELinux では旧カーネルと比較して，Create を除くすべての項目で性能が向上している．特に read/write においては，処理時間増加率が 157%から 29%に低下している．性能が向上した要因として，dentry_open フック関数の実装があげられる．dentry_open 関数を用いることで，すでに open しているファイル等に対して，ポリシ探索の回数を抑制することができる．また，0 KB ファイルの削除においても，処理時間増加率が 80%から 12%に低下している．この要因として，sock_rcv_skb フック関数の性能向上が考えられる．一方で，stat，read/write，0 KB ファイルの作成では依然として 3 つのセキュア OS の中で，処理時間増加率が最も高い．SELinux はラベル方式で資源を識別するため，inode_init_security フック関数や inode_create フック関数を用いてラベルの初期化や設定を行う必要があり，このことが性能低下につながっていると考えられる．

TOMOYO Linux では，旧カーネルと比較して，ファイルの生成，削除処理で大きく処

表 3　Linux カーネル 2.6.30 における LMbench のファイル操作時の処理時間と増加率（単位：μs）
Table 3　Processing time of file operations and its increase rate in Linux kernel 2.6.30 measured by LMbench (unit: μs).

	Normal	SELinux		TOMOYO		LIDS	
stat	1.79	2.67	(48%)	1.83	(2%)	2.20	(22%)
open/close	2.74	3.94	(44%)	4.78	(75%)	3.28	(20%)
read/write	0.37	0.47	(29%)	0.37	(0%)	0.37	(0%)
0 KBfile Create	15.16	58.18	(283%)	53.58	(253%)	16.84	(11%)
0 KBfile Delete	8.20	9.26	(12%)	33.10	(303%)	9.91	(21%)
10 KBfile Create	47.84	89.38	(87%)	83.32	(74%)	48.42	(1%)
10 KBfile Delete	20.06	20.18	(0.6%)	42.68	(112%)	21.16	(5%)

表 4　Linux カーネル 2.6.19 における LMbench のファイル操作時の処理時間と増加率（単位：μs）
Table 4　Processing time of file operations and its increase rate in Linux kernel 2.6.19 measured by LMbench (unit: μs).

	Normal	SELinux		TOMOYO		LIDS	
stat	2.64	4.34	(64%)	2.63	(0%)	2.79	(6%)
open/close	3.98	6.45	(62%)	9.33	(134%)	3.96	(0%)
read/write	0.58	1.49	(157%)	0.58	(−1%)	0.58	(−1%)
0 KBfile Create	11.76	42.80	(264%)	16.36	(39%)	14.00	(19%)
0 KBfile Delete	6.25	11.22	(80%)	9.61	(54%)	6.67	(7%)
10 KBfile Create	34.34	69.26	(102%)	37.78	(10%)	35.48	(3%)
10 KBfile Delete	16.14	19.28	(19%)	19.02	(18%)	16.90	(5%)

理速度が低下している．性能低下の要因としては，パス名の取得と削除における実装の変更があげられる．詳細については 4.3.2 項で述べる．一方で open/close に関しては，処理時間増加率が改善されている．旧カーネルでは，パス名の取得に inode_permission を用いていたが，新カーネルでは，dentry_open を用いており，処理時間やポリシ探索の回数が減ったことが性能改善の要因と考えられる．

LIDS では，旧カーネルと比較して複数の項目で処理時間増加率が大きくなっている．しかし，極端に処理時間が増加している項目は存在せず，他の 2 つのセキュア OS に比べファイルの生成，削除における処理時間は特に短い．

4.3.2　個別のシステムコールのオーバヘッド詳細評価

（評価 1）の結果より，新旧カーネル間で性能が特に大きく変化した項目に関してその要因を詳しく分析するため，新旧カーネル間での以下の評価を行った．

（評価 2-1）SELinux のファイル削除処理（close システムコール）

（評価 2-2）TOMOYO Linux のファイル生成処理（creat システムコール）

表 5　Linux カーネル間における SELinux での close システムコール（単位：μs）
Table 5　Close system call in defferent version of SELinux kernel (unit: μs).

2.6.30				2.6.19			
hookname	sum	count	ave	hookname	sum	count	ave
file_free	183,043.630	1,370,941	0.133	file_free	166,246.612	1,591,046	0.104
inode_free	2,105.578	10,458	0.201	inode_free	2,380.437	14,669	0.162
inode_delete	97.809	973	0.100	inode_delete	62.058	1,074	0.058
sock_rcv_skb	989.218	8,253	0.120	sock_rcv_skb	11,027.765	14,583	0.756
sk_free	1,617.659	5,821	0.278	task_free	3.907	31	0.126
cred_free	10.074	92	0.109				
		average sum	0.942			average sum	1.206

表 6　Linux カーネル間における TOMOYO Linux での creat システムコール（単位：μs）
Table 6　Creat system call in different version of TOMOYO Linux kernel (unit: μs).

2.6.30				2.6.19			
hookname	sum	count	ave	hookname	sum	count	ave
inode_permission	127,585.395	3,588,000	0.035	inode_permission	366,962.831	8,124,000	0.045
file_alloc	31,367.404	897,000	0.035	file_alloc	77,799.981	2,031,000	0.038
inode_create	31,240.724	897,000	0.035	inode_create	5,971,299.965	2,031,000	2.940
inode_alloc	31,425.973	897,000	0.035	inode_alloc	72,487.431	2,031,000	0.036
inode_init_security	33,046.422	897,000	0.035	inode_init_security	74,019.938	2,031,000	0.036
d_instantiate	32,385.181	898,000	0.036	d_instantiate	72,418.537	2,031,000	0.035
path_mknod	12,207,508.304	897,000	13.609	task_free	2.593	8	0.324
dentry_open	22,372,903.197	897,000	24.942	task_kill	0.402	2	0.201
cred_free	0.138	2	0.069				
		average sum	38.830			average sum	3.655

LSMPMON のシステムコール情報を利用し，対象となるシステムコール処理時に呼び出される LSM フック関数の処理時間（μs）を評価した．これまでの評価と同様に，LMbench を 5 回実行した．

（評価 2-1）の結果を**表 5** に示す．表中の sum は各 LSM フック関数の合計処理時間，count は呼び出し回数，ave は平均処理時間を表す．新カーネルでは旧カーネルに比べて，多くの LSM フック関数において処理時間が増加している．一方で，sock_rcv_skb フック関数の処理時間は性能チューニングにより，大幅に短くなっている．旧カーネルでは，この LSM フック関数が大きなオーバヘッドになっており，新カーネルの SELinux では，この LSM フック関数の処理速度が改善されたことで，全体としても処理時間が短くなっている．

（評価 2-2）の結果を**表 6** に示す．なお，実装されていないフック関数が呼ばれた場合でも，LSMPMON では空の関数を呼び出す時間を測定する．表 6 では，新カーネルの path_mknod，dentry_open，旧カーネルの inode_permission，inode_create，task_free が実装されているフック関数であり，その他のフック関数は実装されていない．新カーネルでオーバヘッドとなるフック関数は path_mknod と dentry_open，旧カーネルでオーバヘッドとなるフック関数は inode_create である．inode_create と path_mknod はどちらもパス名を取得するた

めの LSM フック関数である．旧カーネルでは，inode_create では相対パスしか取得できないため，絶対パスの取得に独自実装を用いていた．新カーネルでは，path_mknod によりパス名をルートディレクトリからの絶対パスで取得している．これによって LSM フック関数のオーバヘッドが大きくなっている．また，dentry_open はポリシ探索の回数を減らすための LSM フック関数であるが，この関数自体も大きなオーバヘッドになっている．パス名取得のためのフック関数が実装されたことで，creat システムコール処理時のオーバヘッドが大きくなっていることが分かる．

以上の評価結果から，以下のことが分かった．

(1) SELinux では，特定の LSM フック関数の処理時間増加率の低下により，ファイルの削除におけるオーバヘッドが低下している．また，ファイル生成を除くすべての項目で処理時間増加率が低下しており性能が改善されているといえる．

(2) TOMOYO Linux では，ファイルの生成，削除において処理時間増加率が非常に大きい．これは，アクセス制御の単位であるパス名を求める機能が，独自実装のものから LSM フック関数へと実装が変わったためである．また，open/close については処理時間増加率が低下している．

(3) LIDS では，全体的に処理時間増加率が大きくなっているが，ファイル生成や削除の項目では，依然として 3 つの OS の中で最も小さい．

4.4　セキュア OS の導入による性能への影響の評価

Linux カーネル 2.6.30 へのセキュア OS の導入について表 3 から以下のことが分かる．

SELinux では，stat，read/write，0 KB ファイルの作成の処理時間増加率が，3 つのセキュア OS の中で最も高い．また，他の項目でも，比較的に処理時間増加率が高い．このことから，ファイル処理においてはオーバヘッドが大きいと考えられる．

TOMOYO Linux では，stat の処理時間は 3 つのセキュア OS の中で最も短い．一方で，ファイルの生成，削除処理では特にオーバヘッドが大きい．また，open/close においても比較的オーバヘッドが大きい．これらのことから，ファイルの生成等を繰り返すメールサーバでは特にオーバヘッドが大きくなる可能性がある．

LIDS では，他の 2 つのセキュア OS に比べファイルの生成，削除における処理時間が短く，他の項目においても処理時間増加率は小さい．このことから，ファイル処理に最も適していると考えられる．

4.5　実装方式の違いによる特徴

（評価 3）の結果を**表 7** に示す．なお，表 7 中の “N/A” は，対象の LSM フック関数内で，各セキュア OS のセキュリティ機能が実装されていないことを表す．SELinux と LIDS では i ノードを基にした LSM フック関数を，TOMOYO Linux ではパス名を基にしたフック関数を用いている．

表 7　各ファイル操作時に呼び出される LSM フック関数の処理時間（単位：μs）
Table 7　Processing time of LSM hooks called in each file operation (unit: μs).

hookname	No_secure	SELinux	TOMOYO	LIDS
path_mknod	0.045	N/A	13.609	N/A
path_unlink	0.035	N/A	23.164	N/A
path_truncate	0.060	N/A	13.426	N/A
inode_init_security	0.036	20.658	N/A	N/A
inode_create	0.035	19.037	N/A	N/A
inode_unlink	0.036	0.216	N/A	0.127
inode_permission	0.035	0.159	N/A	0.192
dentry_open	0.041	0.222	17.381	N/A

SELinux では，inode_create と inode_security の処理時間が特に長い．これは，新たに作成された inode に対してラベルの再計算とその初期化をそれぞれ行っているためである．

TOMOYO Linux では path_mknod，path_unlink 等，パス名を扱う関数ではいずれも処理時間が長い．パス名の取得が必要であることや，文字列の比較によってパーミッションのチェックを行っていることが原因として考えられる．

LIDS では，inode 方式で資源を識別するため，ラベルの計算やパス名の取得を行う LSM フック関数を必要としないため，ファイルの生成と削除においては最もオーバヘッドが小さい．このことがセキュア OS 全体のオーバヘッドが小さいことの要因であると考えられる．

以上の評価結果から，以下の 3 点のことが分かる．

(1) SELinux では，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ヘッドが大きい．他の項目についても比較的オーバヘッドが大きい．

(2) パス名による識別を行う TOMOYO Linux は，パス名の取得，削除が必要な場合のオーバヘッドが特に大きくなる．また，open/close においてもパス名の参照（文字列の比較等）の時間が必要なため低速になっている．

(3) ラベル付けや，パス名の取得が必要のない LIDS はファイル処理においてはオーバヘッドが小さい．

5. おわりに

LSM をベースとしたセキュア OS の性能を詳細に測定できる LSMPMON を Linux カー

ネル 2.6.30 に対応させ，3 つのセキュア OS の性能評価の結果について述べた．評価では，特にファイル操作時に多用される LSM フック関数について評価した．これにより，新たにカーネルのバージョンの違いによるセキュア OS の性能の違いや仕様への影響を明らかにした．

SELinux では旧カーネルと比較して，ほぼすべての項目で性能が向上している．特に read/write においては，処理時間増加率が 157%から 29%に低下している．また，0 KB ファイルの削除においても，処理時間増加率が 80%から 12%に低下している．一方で，stat，read/write，0 KB ファイルの作成では依然として 3 つのセキュア OS の中で，処理時間増加率が最も高い．TOMOYO Linux では，旧カーネルと比較して，ファイルの生成，削除処理で大きく処理速度が低下しており，ファイルの削除におけるオーバヘッドは 3 つのセキュア OS の中で最も大きい．一方で open/close に関しては，処理時間増加率が改善されている．LIDS では，旧カーネルと比較して複数の項目で処理時間増加率が大きくなっている．しかし，極端に処理時間が増加している項目は存在せず，他の 2 つのセキュア OS に比べファイルの生成，削除における処理時間は特に短い．また，従来より新しいカーネルでのセキュア OS 導入による影響を詳細に評価した結果を報告した．

LSMPMON のソースコードは LSMPMON プロジェクトページ[9] で公開している．

参　考　文　献

1) Wright, C., Cowan, C., Morris, J., Smalley, S. and Kroah-Hartman, G.: Linux security modules: General security support for the linux kernel, *Proc. 11th Annual USENIX Security Symposium*, pp.17–31 (2002).
2) 松田直人，佐藤和哉，田端利宏，宗藤誠治：LSM を利用した性能評価機能の実現と評価，電子情報通信学会論文誌 D，Vol.92, No.7, pp.963–974 (2009).
3) NSA: Security-Enhanced Linux, available from ⟨http://www.nsa.gov/selinux/⟩.
4) Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System, *Proc. FREENIX Track: 2001 USENIX Annual Technical Conference*, pp.29–42 (2001).
5) TOMOYO Linux, available from ⟨http://tomoyo.sourceforge.jp/⟩.
6) 原田季栄，半田哲夫，板倉征男：TOMOYO Linux の設計と実装，コンピュータシステム・シンポジウム論文集，Vol.2009, No.13, pp.101–110 (2009).
7) LIDS, available from ⟨http://www.lids.org/⟩.
8) LMbench, available from ⟨http://www.bitmover.com/lmbench/⟩.
9) LSM Performance Monitor, available from ⟨http://www.swlab.cs.okayama-u.ac.jp/lab/yamauchi/lsmpmon/⟩.

(平成 22 年 11 月 30 日受付)
(平成 23 年 3 月 7 日採録)

山本　賢治

2009 年岡山大学工学部情報工学科卒業．2011 年同大学大学院自然科学研究科博士前期課程修了．同年日立ソリューションズ入社．コンピュータセキュリティに興味を持つ．

山内　利宏（正会員）

1998 年九州大学工学部情報工学科卒業．2000 年同大学大学院システム情報科学研究科修士課程修了．2002 年同大学院システム情報科学府博士後期課程修了．2001 年日本学術振興会特別研究員（DC2）．2002 年九州大学大学院システム情報科学研究院助手．2005 年岡山大学大学院自然科学研究科助教授．現在，同准教授．博士（工学）．オペレーティングシステム，コンピュータセキュリティに興味を持つ．電子情報通信学会，ACM，USENIX 各会員．