5 高信頼を実現するLinuxの新しい機能

宗藤誠治（日本アイ・ビー・エム（株）） 須崎有康（（独）産業技術総合研究所）

本稿では，Linux のバージョン 2.6.30 から導入された，IMA（Integrity Measurement Architecture）と呼ばれる完全性情報の計測および保護を行う機能について説明する．これは Trusted Computing を実現する OS レベルの機能であり，セキュリティチップと連動することで高信頼のシステム構築を実現する．

ソフトウェアの完全性計測と Trusted Computing

利用者や社会に大きな被害を及ぼすマルウェアが後を絶たない．マルウェアの侵入や感染を防ぎ，感染した際には迅速な検知と駆除が必要であるが，マルウェアによる改ざんを完全には防げない，また検出できない既存システムが多く存在する．「ソフトウェアの完全性（Integrity）」とはそうした改ざんが存在しないことを目指すものである．

現在のソフトウェアの完全性へのアプローチは導入時と運用時のチェックが主流である．まず，導入時ではコード署名を用いた由来の確認を実施する．ソフトウェアを導入する前に，ソフトウェアの発行元を署名により識別し，不正なソフトウェアや改ざんされたソフトウェアであるか検証することができる．Microsoft 社の Autheniticode や，Linux のパッケージ管理システムなど主要な OS ではこうした検証を実施している．

導入済みのソフトウェアやファイルシステムなどが不正に改ざんされてしまう可能性がある場合は，Tripwire や AIDE などのファイルシステムの完全性検証ツールを利用してファイルシステムを検査し，定期的にその記録と最新のファイルシステムとの整合性を確認するなど運用時の定期的な検証が必要となる．しかしながら，マルウェアの高度化による検知の難しさや，対策ソフト自身が無効化されるような攻撃など，限界が見えてきている．また，利用者にも大きな負担を強いている．

最新の Linux Kernel は IMA（Integrity Measurement Architecture）と呼ばれる，完全性計測および保護機能を OS カーネルレベルでサポートしている．完全性計測ではファイルやそのメタデータに改ざんがないことを確認するためにそのハッシュ値を計測する．この背景にあるのが，Trusted Computing という新しい計算機アーキテクチャであり，ハッシュ値の保存に耐タンパーなハードウェアを使った「信頼のよりどころ（Root of Trust）」を利用することで，シンプルで強固なソフトウェアの完全性保護の実現を目指している．

本稿では，まず IMA の背景となっている Trusted Computing に関連した技術として，セキュリティチップ（Trusted Platform Module：TPM），およびソフトウェアの起動シーケンスを記録する Trusted Boot について簡単に解説する．その後，IMA の詳細を解説し，最後に課題と今後の方向性について説明する．

■ TPM の基本的な機能

TPM とは Trusted Computing を実現するために開発されたセキュリティチップの一種である．すでに

多くの PC やサーバに搭載されており，鍵管理，暗号処理，乱数生成，安全な完全性情報の保存，ストレージの提供，完全性情報の報告機能などを持つ．

従来のセキュリティハードウェアは非常に高価であり，PC などの民生機器での利用が難しかった．そのため，安価だが必要最小限のセキュリティ機能を提供するデバイスとして，TPM が提案された．セキュリティ技術を安価に利用するためには，その技術の標準化も重要である．そこで 1999 年に業界標準化団体（TCPA のちに TCG[4]）が結成され，関連する技術の標準化と製品化が進められることになった．

また，TPM を安価に実現するため，その機能は従来のセキュリティチップやモジュールとは次のように異なる．

1）TPM は計測を行わない

TPM は受動的なデバイスであり，それを利用するソフトウェアがすべてを制御しなければならない．

2）TPM はセキュリティアクセラレータではない

TPM は RSA やハッシュ関数の計算機能を持つが，暗号処理能力はスマートカード程度しかない．チップが接続されるバスも非常に低速である．そのためハッシュの計算などは本体の CPU を使った方が遥かに高速である．ただ，結果のダイジェスト値は TPM に保存し保護する．RSA 鍵を使った演算は TPM チップで行うが，処理時間がかかるため，高頻度で RSA 演算を使用する用途に TPM は向かない．

3）TPM への物理攻撃は可能である

TPM は耐タンパー機能を持つが，サイドチャネルアタックや，分解による TPM 内部の RSA 鍵の取得の報告がされているように，物理攻撃に対する耐性には限界がある．ただ，ISO/IEC 15408 や FIPS 140 などのセキュリティ評価により，利用者は利用している TPM の持つセキュリティ機能について知ることができる． また，TPM の利用の主な目的は実際に危険が多く大規模な被害につながりやすいソフトウェアレベルの攻撃への対策である．

TPM の RSA 鍵，保存レジスタ，完全性保護で利用する主な命令を表 -1 に示す．TPM やその活用について詳しく知りたい場合は文献 2），3）を参考にしていただきたい．まとめると，TPM は完全性計測を安全に保存する場所（PCR）とレポートする機能（Quote）や利用する機能（Seal/Unseal）を提供する．これはソフトウェアのみの完全性保護では困難な機能であり，TPM とソフトウェアが連携することで，高信頼なシステムを実現することが可能となる．

■ Trusted Boot

次に，TPM を利用して実際にどのようにソフトウェアの完全性保護を行うかを説明する． これまでのソフトウェアの完全性の確認は，逐次的に実施する場合が多かったが，Trusted Boot ではソフトウェアの完全性の計測と確認を分離する一括（POST）確認方式が可能である．図 -1 に示すように，従来は次に起動するソフトウェアを計測し，期待値と比較してその起動の可否を決定する．この期待値や署名の検証のための証明書などは事前に配布される．ソフトウェアの導入確認も同じアプローチである．一方，Trusted Boot では計測のみ行い，計測結果は安全な TPM に保存する．完全性の確認は任意のタイミングで実施可能であり，柔軟なシステム運用を可能にする．Trusted Computing では計測された完全性情報がその後のソフトウェアによって改ざんされることがないため，このような一括（POST）確認方式のアプローチが可能となる． ただ，最初に述べたように TPM 自体は計測を行わないため，最初に起動するコード 1 の完全性を計測し確認する方法が問題になる．これについては最新の PC Platform（x86 アーキテクチャ）では次の 2 種類の「信頼のよりどころ（Root of Trust）」を構成する方法が利用できる．

SRTM（Static Root of Trust Measurement）

TPM は受動的なデバイスなので，最初に起動するコードは計測できない．そこで，最初に起動するコードを書き込み禁止の領域に保存する．PC の場合は BIOS の中で最初に起動する部分であり，CRTM（Core Root of Trust Measurement）と呼んでいる．結果として，PC の Root of Trust は TPM と CRTM で構成されることになる．CRTM は自身の

機能	説明
EK *Endorsement Key*	TPM 固有の 2046 bit RSA 鍵である． TPM ベンダや PC ベンダがこの鍵に対して証明書を発行することで，第三者が TPM の正当性を検証することができる．
SRK *Storage Root Key*	2046 bit の RSA 鍵で利用者が TPM を初期化（TakeOwnership）した際に生成される． TPM 内部で保存される RSA 鍵は，EK と SRK のみであり，TPM で利用するそのほかの各種 RSA 鍵は，SRK で暗号化されて TPM の外部で保存される．鍵をロードする手間がかかるが，外部で保存することにより，TPM で利用する鍵の数の制約はなくなっている．
AIK *Attestation Identity Key*	2046 bit の RSA 署名鍵である． 普通の RSA 署名鍵との違いは，AIK が TPM 内部で利用する鍵であることを第三者（PrivacyCA）が確認し，証明書を発行できるようになっていることで，TPM には AIK の証明書発行のために特殊な命令を備えている． もし，EK を使い署名を行うと TPM が特定されるため，プライバシーへの配慮から，完全性情報に TPM を使って署名を施す場合には AIK を用いる．EK は AIK の生成と証明書発行にのみ利用され，EK を用いた署名はできない．
PCR *Platform Configuration Register*	計測したハッシュ値を記録するための 20 バイトのレジスタで，v1.2 の TPM では 24 個存在する．PCR は TPM のリセットとともに初期化され，以下の TPM_Extend と呼ばれる操作でのみ更新が可能である． この PCR の値を AIK を使い TPM 内で署名することで TPM やプラットフォームの外からでも確実に PCR の値を確認することができる．これは Remote Attestation と呼ばれている．ただ，PCR の値や，対応するイベントログの収集および検証は複雑な作業である．そこで TCG では Platform Trust Services（PTS）と呼ぶ検証のためのコンポーネントの仕様の策定を進めている． また，PCR 値がある特定の値のときにのみ，RSA 鍵や TPM 内部のリソースが利用できるように設定することも可能である．たとえば，TPM の Seal，Unseal 関数を用いると，特定の PCR 値のときにのみ，データが復号化できるようになる．Microsoft のディスク暗号化機能，BitLocker では，正しい起動が行われた際の PCR 値を使って，HDD 暗号化鍵の保護を実現している．たとえば，外部 CD で PC を起動した場合には起動にかかわるコンポーネントが異なるため，PCR 値は通常とは異なり，HDD にはアクセスできない．また，暗号化された HDD を別の同じ BIOS のマシンで起動して PCR 値を同じにしても，TPM が異なるために Seal 鍵が利用できず，起動はできない．つまり，PC（TPM）と PCR 値が整合しないと，HDD の中身にはアクセスできない．これは，非常に便利な機能だが，PCR 値を一意に管理できないと利用できない．IMA の場合は，絶えず PCR 値が変化するため，IMA が利用する PCR を Seal/Unseal で利用することは難しい． どの PCR を誰が利用するかは，仕様によって定められている．たとえば，PC では PCR の 0-7 は BIOS とブートローダが利用している．標準構成の IMA は PCR の 10 を利用する．PCR の 17-22 は次に述べる DRTM で利用される．
TPM_Extend	TPM の命令．PCR の値を変更する．新しい PCR の値は以下のようになる． PCR＝SHA1（現在の PCR 値 ＋ 記録したいハッシュ値） PCR の数には限りがあるため，1 つの PCR に複数の完全性計測を割り当てる必要がある．TPM_Extend による更新の結果，PCR 値は一連の記録を反映した値を示す． PCR へ記録する対象と順序が一意に決まっているなら，PCR 値そのものを用いて，その一連の計測を確認することが可能である．また，PCR へ記録する対象と順序が変化する場合，PCR 値自体で完全性を確認することはできないが，PCR への記録のログがあれば，このログを用いて PCR 値を再現させることができる．つまり，ログの完全性を PCR で証明することができる．こうしたログはイベントログと呼ばれ，実際の Platform の検証では，ログの記録を元 に完全性の確認を行い，ログ自体の完全性の確認に PCR を用いる．
TPM_Quote	TPM の命令．AIK を用いて PCR 値に署名を施す．
TPM_Seal **TPM_Unseal**	TPM の命令．暗号化，復号化を行うが，設定した PCR 値の時にのみ復号化が可能．OS などが使う暗号鍵の保護に利用されている．

表 -1　完全性計測と保護，報告に関係する TPM の主要機能

計測と，次に処理を移す BIOS のコードを計測し，TPM に記録する．BIOS は Option ROM や設定情報などの計測を行った後ブートローダを計測し，ブートローダに処理を移す．ブートローダは BIOS と連携して OS の計測を行い，OS に処理を移す．OS で IMA が有効な場合は，IMA の計測は Root of Trust から始まる一連のチェインでその完全性を検証できるようになる．この BIOS による Root of Trust は SRTM と呼ばれる．

DRTM（Dynamic Root of Trust Measurement）

残念ながら BIOS の品質は十分とはいえず，SRTM で計測ができていたとしても，そのセキュリティ機能は保障されていない．そこで，Static な起動を担う BIOS に依存せずに，Root of Trust を実現する方法として，CPU とチップセットによるソフトウェアに依存しない計測方法として DRTM が実現された．DRTM では起動後の任意のタイミングでダイナミックに，Root of Trust を実現すること

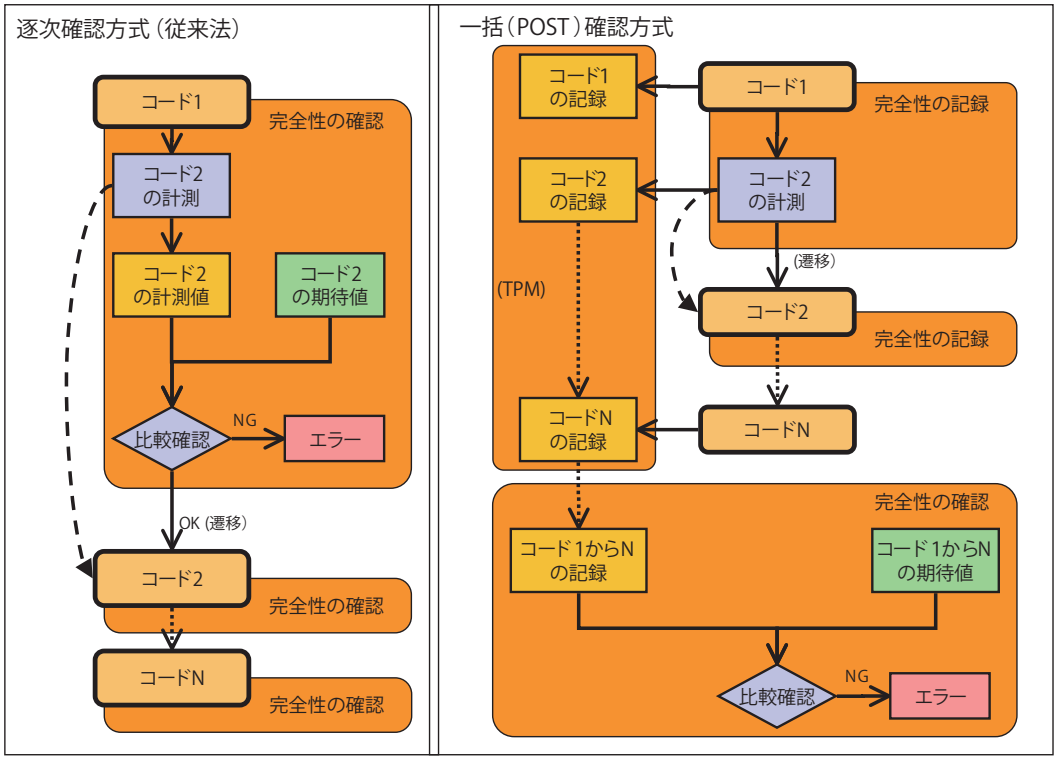

図 -1　完全性の計測と確認のタイミング

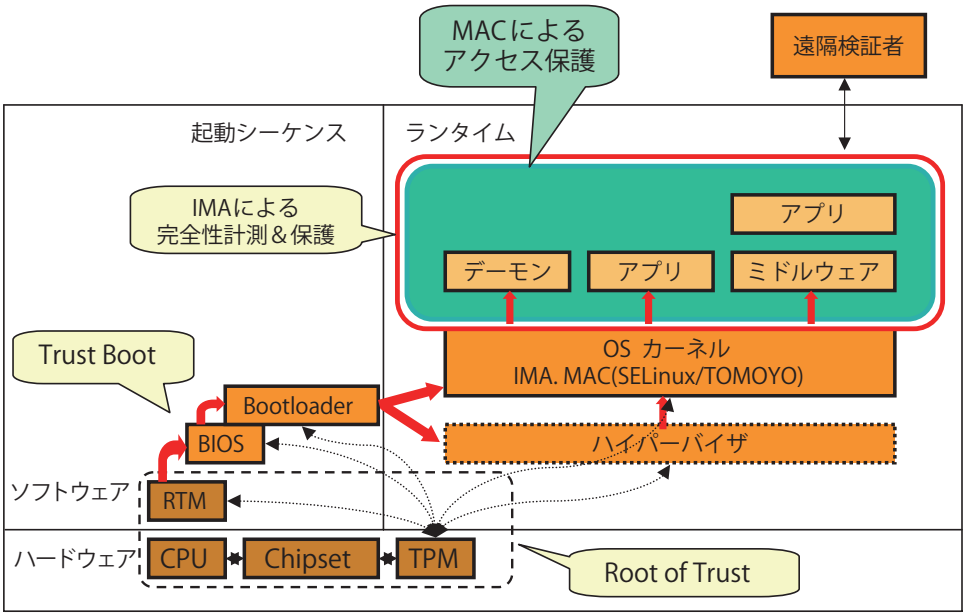

図 -2　IMA と MAC で実現する高信頼 Linux

ができる．この DRTM を基点にした Trusted Boot によってハイパーバイザや OS を起動することで，SRTM よりも柔軟な完全性の保護を実現できる．

図 -2 の左下に Trusted Boot の概要を示す．結果として OS 環境の完全性が計測され，そこで実行される強制アクセス制御（Mandatory Access Control : MAC）や IMA などのセキュリティ機能が正しく稼働していることを確認できるようになる．Trusted Boot 実現のためには，TPM とともに BIOS やブートローダなど起動で利用するコンポーネントの対応が必要となる．

以上，IMA の前提となる TPM と Trusted Boot に

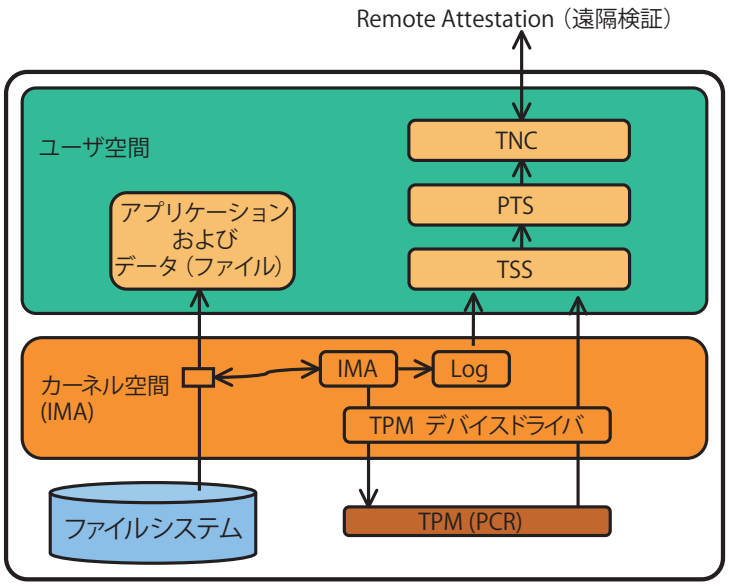

図 -3　IMA の構成

ついて説明を行った．次に，IMA の提供する機能について説明する．

IMA（Integrity Measurement Architecture）

Linux では IMA と呼ばれている完全性情報の保護機能がバージョン 2.6.30 からサポートされている．

ここで言う完全性情報とは，ファイルやそのメタデータに改ざんがないことを確認するための情報であり，ファイルやメタデータのハッシュ値となる．IMA はこれらを OS レベルで記録する．IMA 自体は TPM がなくても利用可能であるが，前述の Trusted Boot と組み合わせることで，TPM を礎にした信頼の鎖をシステム全体で構築する．IBM の OS である AIX でも IMA と似た機能の Trusted Execution をサポートするが，ハードウェアベースの保護機能と連動した高信頼なシステムの構築が可能となったのは Linux の IMA が最初である．

■メインライン化への過程

IMA についての最初の論文発表は 2004 年の Usenix Security[1] である．その後，翌年の 5 月に LKML（Linux Kernel Mailing List）へパッチが提出され，実装が一般に公開された[5]．当時の IMA は LSM（Linux Security Module）が持つスタック機能を用いて実装されていた．これは SELinux などの MAC との同時使用を念頭においていたためである．この提案は LSM をアクセス制御のみに使うべきだとの理由で正式にリジェクトされた．LSM についてはそのほかにもさまざまな議論があり，LSM のスタック機能が廃止されたことで IMA と MAC と共存ができない状態が続くことになる．IMA の開発者たちはその後も継続的にパッチを提出し，2007 年に完全性保護のためのインタフェースを LSM と分離した LIM（Linux Integrity Module）を提案した．最終的には，2009 年 6 月のバージョン 2.6.30 で IMA はメインライン化されたが，IMA の実装はまだ実験的であり，その後も開発者による機能の修正と拡張が続けられている．IMA に限らないが，Linux に新しい機能を加えるには，長い議論と忍耐，そして長期間開発者を支える組織が重要である．

■ IMA の動作

図 -3 に IMA の構成を示す．IMA はカーネル空間で動作し，メモリ上に展開される実行ファイルやデータ，カーネルモジュールなどを計測する．計測は TPM に記録され，イベントログとしてカーネル内で保存される．計測された結果は TSS（TCG Software Stack，TPM の機能を補完するサービスデーモン），PTS（Platform Trust Service，完全性情報の収集と検証を行うコンポーネント）を介して TNC（Trusted Network Connect，検疫ネットワーク）などで利用される．たとえば，VPN 接続時に完全性を検証し，OK であれば接続を許可し，NG の場合は問題の修正方法を提示して検疫 LAN に接続し修正を促す[3]．

初期の IMA はカーネルモジュールと実行ファイルのみを計測したが，現在は任意のファイルの計測

デフォルトの計測ポリシー	dont_measure fsmagic=PROC_SUPER_MAGIC dont_measure fsmagic=SYSFS_MAGIC dont_measure fsmagic=DEBUGFS_MAGIC dont_measure fsmagic=TMPFS_MAGIC dont_measure fsmagic=SECURITYFS_MAGIC dont_measure fsmagic=SELINUX_MAGIC measure func=BPRM_CHECK measure func=FILE_MMAP mask=MAY_EXEC measure func=PATH_CHECK mask=MAY_READ uid=0
SELinux のラベルを用いた計測ポリシーの例	dont_measure obj_type=var_log_t dont_measure obj_type=auditd_log_t
SMAK のラベルを用いた計測ポリシーの例	measure subj_user=_ func=INODE_PERM mask=MAY_READ

表 -2　IMA の計測ポリシー [5)]

が行えるように拡張されている．単純に OS が開くすべてのファイルを計測すると，性能への影響が大きく，その検証コストも増大するため実用性に問題がある．計測対象としては，OS を構成する基本部分である TCB（Trusted Computing Base）の保護が重要であるが，Linux では TCB の境界は明確ではない．**表 -2** に示すように IMA ではポリシーにより計測を制御している．デフォルトの計測ポリシーではすべての実行形式ファイル（ライブラリも含む），メモリマップされた実行ファイル，管理者（Root）権限で開かれたファイルの計測と記録を行う．ファイルシステムのタイプによる計測の制御が可能であるが，SELinux や SMAK のようにラベルによる MAC と併用する場合には，ラベルによって計測を制御することも可能である．こうして計測された情報は，拡張アトリビュートの security.ima に保存されている．

ファイルの計測管理にはテンプレートを用いている．現在（Version 2.6.32），テンプレートに含まれる情報はファイル名とファイルのハッシュ値（SHA1）である．このテンプレートのハッシュ値が TPM に記録され，1 ファイルにつき，1 つのイベントが対応することになる．現在，そのほかのメタデータの計測についても機能の拡張が検討されており，この記事が読まれている頃には SHA1 以外のハッシュアルゴリズムへの対応と，セキュリティラベルの計測が実現しているかもしれない．

また，IMA が記録する最初のイベントは Boot Aggregate である．これはブートローダから OS に制御が移った直後（IMA 起動時）の PCR0 から PCR7 の値から生成された値で，これにより OS 起動時の環境と IMA のログが一意に対応付けられる．

まとめると，IMA は以下のように動作する．

1）コードの実行やファイルのオープンの前に呼び出される．
2）対象となるファイルのハッシュ値を計算する．ハッシュ値計算には時間がかかるため，対象のファイルはその間ロックされる．
3）テンプレートを作成し，PCR に記録するテンプレートのハッシュ値を計算する．
4）TPM にハッシュ値を記録し，イベントログを更新する．
5）ファイルの実行を行う．もしくはアプリケーションにファイルを渡す．

計測したプログラムにバッファオーバフローで「外部コード」が実行される脆弱性があるなら，外部コードは現在の IMA では計測できない．ただし，そうした脆弱性が既知であるなら IMA を用いて，脆弱なプログラムを特定することは可能である．いずれにせよ IMA は，計測の後にメモリ上にロードされたプログラムが変更されない，また振舞いが変わらないことが前提である．

IMA では「読み出し」で開かれているファイルへの「書き込み」や，「書き込み」で開かれているファイルへの「読み出し」で発生する Time-of-measurement

フック	機能
ima_inode_alloc	計測保存領域の確保
ima_inode_free	計測保存領域の開放
ima_bprm_check	実行ファイルの計測
ima_file_mmap	mmap された実行ファイルの計測
ima_file_check	ファイルを計測，カウンタをインクリメント
ima_file_free	ファイルの開放，カウンタをデクリメント
ima_count_get	ファイルの読み書きを監視（ToMToU 対策）

表 -3　IMA のフック一覧（Kernel 2.6.34）

Time-of-use（ToMToU）と呼ぶレースコンディション対策として，カウンタを用いた計測の整合性チェックを実施している． なお，IMA の詳細な動作については，Linux のカーネルソースの security/integrity/ima/ 以下にあるソースコードを参照されたい．IMA では表 -3 に示す 7 つのフックを用いて完全性計測を実現している．

■ EVM（Extended Verification Module）

IMA は完全性計測の基本機能を提供しているが，IMA を利用する新しいセキュリティ機能として EVM（Extended Verification Module）が LKML に提案されており，メインライン化に向けた活動が進められている．この EVM は強制アクセス制御モジュールの一種で，類似の機能としては，digsig がある．digsig ではファイルの署名を元に強制アクセス制御を行うが，EVM は IMA によるファイルのメタデータ計測を元に，強制アクセス制御を実施する．情報は拡張アトリビュートの security.evm に保存され，HMAC のキーは TPM で暗号化され，OS が期待した手順で起動したときのみ利用可能となる．

EVM の利用方法としては，SELinux などで利用されるセキュリティラベルの保護がある．たとえば，システムがオフラインになり，ファイルシステムがほかのシステムでマウントされた際には本来の OS が提供していた保護機能はバイパスされ，MAC の前提となるセキュリティラベルの改ざんも容易である．再びオンラインになった際には，改ざんされたセキュリティラベルによって MAC が実行されることになる．EVM は，こうした改ざんを確実に検出可能である．

■ IMA と強制アクセス制御（MAC）の連携

IMA と MAC の関係を図 -2 右側に示す．IMA と MAC は異なるセキュリティ機能を提供しているが，それらは相補的である．MAC はラベルやパスを元にプログラムの挙動をポリシーで規定し，違反する挙動を阻止するとともに，記録する．IMA（や EVM）はポリシーやラベルの完全性を保障することで，MAC によるセキュリティ保護をより完璧にする．つまり，Linux は非常に高信頼なプラットフォームが実現可能な OS であるといえる．

■ 従来技術との比較

よく，情報セキュリティの三大基本理念として，「機密性（Confidentiality）」「完全性（Integrity）」「可用性（Availability）」の 3 つ（CIA）が取り上げられるが，IMA はこの完全性に特化した OS 機能と言える．完全性を確認するための従来技術としては，Tripwire や AIDE が広く利用されている．こうしたツールは各種ファイルのハッシュ値やメタデータの記録を行いデータベースとして保存する．こうして保存された情報と実際のファイルを定期的に現状と照合することで，不正な改ざんを検出することが可能である．こうした既存技術にはいくつかの問題点がある．

- 作成したデータベース（完全性情報）の安全な保存と利用が必要となる．実際は管理者に依存する．
- 計測や確認作業の負荷が大きい．
- 改ざんの発生と検出の間の時間的なギャップがある．このギャップを小さくするためには，確認頻度を上げる必要があり，運用負荷とセキュリティの確保の間にトレードオフがある．
- ツールを別途管理者権限で稼働しなければいけ

ファイル名	説明
/sys/class/misc/tpm0/device/pcrs	現在の PCR 値
/sys/kernel/security/tpm0/ascii_bios_measurements	BIOS とブートローダのイベントログ（アスキー形式）
/sys/kernel/security/tpm0/binary_bios_measurements	BIOS とブートローダのイベントログ（バイナリ形式）
/sys/kernel/security/ima/ascii_runtime_measurements	IMA のイベントログ（アスキー形式）
/sys/kernel/security/ima/binary_runtime_measurements	IMA のイベントログ（バイナリ形式）

表 -4　TPM，IMA に関連する sysfs ファイル一覧

```
Index PCR   Type        Digest                                   EventData
-------------------------------------------------------------------------------------------------------
   0    0 0x00000008 4081b13dc986e581d587aa7fe6c61e02ef7312b2 [BIOS:EV_S_CRTM_VERSION]
   1    0 0x00000001 8b5c22ae675ea440e2f403b4d5e88131fecc2a1c [BIOS:EV_POST_CODE(EV_CODE_NOCERT)]
   2    0 0x00000001 d9f7d65755c884f6c25680d577f348523006eed3 [BIOS:EV_POST_CODE(EV_CODE_NOCERT)]
<snip>
 198    8 0x00001205 a73c31a3abe77960ff8c631edf0dfd29112aa6f3 [GRUB:KERNEL /boot/vmlinuz-2.6.30-ima]
 199    8 0x00001305 f8422432a897cd434fa405e6c653ad12b6faacd6 [GRUB:INITRD /boot/initrd.img-2.6.30-ima]
 200    8 0x00000004 2431ed60130faeaf3a045f21963f71cacd46a029 [GRUB:EV_SEPARATOR, OS Event Separator]
 201    8 0x00001005 fac33a1fc0ad42c07d00322d64c23f67567f334a [GRUB:ACTION, Booting Big Linux Kenrel]
 202   10 0x00000000 173d86589890c0c8b9b42b5b1ee671aecfbec7c0 [IMA:boot_aggregate]
 203   10 0x00000000 8a11aa2017bfdf52ae1ab8cfb277fc651bc7d611 [IMA:/init]
 204   10 0x00000000 a078e19e5ea2bf75ed353fc6613f7132863618d5 [IMA:/init]
<snip>
 523   10 0x00000000 c5400a3dbdb4ec0e9814498095841fc4ccea0245 [IMA:/sbin/load_policy]
 524   10 0x00000000 b6de25749576ec1fe0b6e49c3394b1d97c737354 [IMA:ld-2.9.so]
 525   10 0x00000000 6935ae5654a23a6c773bbb2b2be05dc268f4d905 [IMA:ld.so.cache]
 526   10 0x00000000 62cba99abf93df49aaa59bf286e95e35a0f54f38 [IMA:libsepol.so.1]
 527   10 0x00000000 4831ec4abdaaa3d8e501fb1b87b9af8e4967d1bd [IMA:libc-2.9.so]
 528   10 0x00000000 6eaf6627796fb433c0fabc0ce92d7c143a0bf50a [IMA:libdl-2.9.so]
 529   10 0x00000000 3e362a6e51956838b20007340c000152b397aa5e [IMA:config]
 530   10 0x00000000 e1187509a3a9b439ee480bce606991178112de0e [IMA:policy.23]
<snip>
```


図 -4　計測された完全性情報の例

ない．

これに対して，Linux の IMA では OS カーネルで完全性情報の計測と記録機能を提供することで，

- 実際にアクセスしたファイルの完全性を記録（逆にシステムが利用していないファイルは未計測になる）．
- 改ざんをリアルタイムに検出可能．
- OS カーネルの機能のため，別途管理者権限で稼働する計測プログラムは不要．

となり，従来とは異なる効率的な完全性情報の計測と保護の仕組みを提供している．

■最新の IMA を使ってみよう

カーネルのメインラインでサポートされたことで，以前に比べると簡単に IMA を利用することができるようになった．IMA は TPM がなくても利用可能であるが，TPM 搭載の PC の場合では TPM を有効にすることで，TPM と連携した記録が実現できる．Trusted Boot を実現するには，GRUB へのパッチの適応や，tboot などの設定が必要であり，その設定については文献 7）を参照されたい．実際に，Fedora 12 では IMA サポートのカーネルが配布されており，Kernel Option に ima_tcb=1 を追加することで，IMA を Enable することができる． BIOS や IMA の計測ログは**表 -4** で示すファイルに securityfs（仮想ファイルシステムの 1 つ）経由で簡単にアクセスできる．**図 -4** にはログの例を示す．

通常，ログには数百から数千のイベントが記録される．図 -4 の例では，計測の 0 からは SRTM に関する計測である．198 は，ブートローダが計測したカーネルのハッシュ値を示す．この例では，SELinux のポリシー読み込みの記録がイベントの 523 から始まっていることが分かる．

■ KNOPPIX を使って IMA を利用

残念ながら IMA や Trusted Boot を利用するため

にはブートローダやOSの再構築が必要である．さらにTPMの設定の難しさもあって開発者以外の利用は難しいのが実情であった．そこで，こうした機能を手軽に利用できる，VMKnoppix[6]を産業技術総合研究所が2008年に作成し一般に公開している．これは1CD Linuxであり，CDから起動することで，現在インストールされているOSを変更することなく，IMAやTrusted Bootを体験できる．VMKnoppixでは複数のLinuxカーネルが起動可能であり，そのうち1つがIMAに対応している．IMAはLinux Kernenel 2.6.19.11にカーネルパッチを当てるもので，現行のものより古いがIMAの基本である計測機能は対応している．IMAの計測結果を参照することで，現在稼働している実行ファイルおよびライブラリの完全性が確認できる．VMKnoppixではさらにTrusted Bootのために GRUB-IMAとTPMドライバを含み，CRTMから始まり，BIOS，Trusted GRUB，Linux IMAに続く信頼の輪(Chain of Trust)も確認することができる．

VMKnoppixは名前が示すように仮想マシンソフトウェアを多数含む1CD Linuxである．含まれる仮想マシンソフトウェアのうちXenとKVMが仮想TPMを提供しており，物理TPMがない環境においてもTrusted Bootを仮想マシン上で確認できる．仮想TPMはスイス連邦工科大学で開発されたTPM Emulatorを利用し，仮想マシンで利用できるように仮想デバイスモデルに追加し，BIOS対応も行ったものである．CDブートで仮想TPMを提供する仮想マシンを立ち上げ，その仮想マシン内にさらにCD内のIMAカーネルを立ち上げるような，ハードディスク依存しない試験環境を提供できる．また，VMKnoppixではこのほかにもTCGソフトウェアスタックであるTrouSerSを含んでおり，TPMを使った鍵生成，ファイルの暗号化など行うことができる．このようにIMAおよびTrusted Computingのための教育教材として手軽な環境のため，大学の教材としても活用されている．

今後に向けて

ようやくLinuxのカーネル機能の一部となったIMAだが，その本格的な利用にはまだ多くの課題が存在する．

■ TPMとIMA普及の現状

現在，多くのPCがTPMを搭載しており，BIOSもTrusted Bootに対応している．また，TPM搭載サーバも増えてきている．主要なLinuxディストリビューションではすでにTPMドライバやTSSは標準でサポートされており，Fedoraのバージョン12以降ではIMAが有効になったKernelも配布されている． IMAについてはまだ連携するコンポーネントとの整合性が不十分であるが，IBMやRed Hatを中心として活発に開発作業が進められている．

Trusted Computingの信頼の根底にあるのは公開鍵暗号によるインフラストラクチャ（PKI）である．しかしながら，このPKIの普及も十分とはいえない．実際，市場に出回っている多くのTPMはEKの証明書を持たない．つまり，証明書によるTPMの信頼性検証ができない． また，AIKの証明書を発行するPrivacyCAサービスも存在しない．

■完全性計測と検証の課題

Trusted BootやIMAによる完全性の計測では，起動やロードの際のみで，その後にメモリ内容の改ざんがあった場合は検出できない．定期的にメモリの内容を計測すること自体は可能であるが，メモリ上にロードされたイメージはファイルとは異なり，セクション単位の計測と管理が必要になってしまう．不正にメモリが変更できないOSを利用し，そのOSが使用されていることを計測により確認するのが現実的である．後で脆弱性が発見された場合には，該当OSであることは計測から確認ができる．

現在，Root Of TrustからVMM, OSまでの起動過程は計測と検証が可能である．たとえば，OpenPTS[7]では起動過程のモデル化を行い効率的な検証を実現している．一方，IMAが計測している

ユーザ空間ではまだ課題が多く，その検証は容易ではない．IMA はホワイトリストに基づく検証と組み合わせることで改ざんの有無が確実に検証できる．つまり，確実なマルウェアの確実な検知が可能になる．ただ，こうした仕組みを実現するためには，多くのソフトウェアの設計変更が必要となるため，本格的な普及には至っていない．今回 Linux では少し実現できたが，既存のソフトウェア資産について，こうした機能を追加するには，さまざまな技術的課題がある．以下に課題を列挙しておく．

- 一般に利用されているデスクトップ OS やサーバ OS では，任意のソフトウェアの導入と構成が可能であり，単純なハッシュ値による検証には限界がある．ただ，組込み機器などでは利用しているコンポーネントを完全に管理することができるため，ホワイトリストを用いた検証や IMA の適応が PC よりも先に進むかもしれない．
- 現状の Linux では TCB の特定も難しく，重要なコンポーネントとそうでないコンポーネントの切り分けができない．
- ユーザ空間で稼働しているソフトウェアには複雑な依存性が存在する．検証にあたってはそうした依存関係も含めて確認する必要がある．
- IMA の計測から OS 上で稼働しているソフトウェアの特定は可能であるが，そのセキュリティ機能が明確でないと，システム全体のセキュリティを推し量ることは難しい．IMA をセキュリティチェックで利用する場合，あるポリシーに照らしあわせて，検査対象が準拠しているか否かを導出したい．
- 長い期間にわたって稼働するシステムではその間ソフトウェアは絶えず更新する．結果として IMA の計測のログには新旧のソフトウェアの記録が残り，サイズもどんどん大きくなってしまう．現状の IMA の計測ではこのようなシステムの検証には不十分であり，今後，システムのライフサイクルを意識した構成管理との連携が必要となる．

以上のように，IMA の本格的な活用にはまだ多くの課題が存在しているが，IMA や EVM を活用した MAC へのオフライン攻撃防御など，すぐに使える機能も存在する．現在，仮想化やクラウドの進展に伴い柔軟かつ安全なセキュリティが求められており，Trusted Computing や，今回紹介した IMA のように，ソフトウェアの完全性を管理し，制御する技術の実用化に期待が集まっている．

参考文献

1) Sailer, R., Zhang, X., Jaeger, T. and van Doorn, L.：Design and Implementation of a TCG-based Integrity Measurement Architecture., In Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA, pp.223-238 (2004).
2) 中村智久，東川淳紀：PC 搭載セキュリティチップ（TPM）の概要と最新動向，情報処理，Vol.47, No.5, pp.473-478 (May 2006).
3) 上杉忠興，圷　毅，宗藤誠治，吉濱佐知子：Trusted Network Connect—TPM の利用管理技術の動向，情報処理，Vol.48, No.11, pp.1232-1241 (Nov. 2007).
4) Trusted Computing Group (TCG)
http://www.trustedcomputinggroup.org/
Trusted Computing を実現する各種仕様が公開されている．
5) Linux-IMA
http://linux-ima.sourceforge.net/
6) VMKnoppix
http://www.rcis.aist.go.jp/project/knoppix/vmknoppix/index-en.html
7) OpenPTS
http://openpts.sourceforge.jp/
IMA の計測を検証するコンポーネント．

（平成 22 年 6 月 30 日受付）

■宗藤誠治（正会員）munetoh@jp.ibm.com

1992 年東北大学大学院電子工学専攻修士課程修了，同年より日本アイ・ビー・エム（株）東京基礎研究所（現職）．

■須崎有康（正会員）k.suzaki@aist.go.jp

東京農工大学大学院中退 (1991)．電子技術総合研究所を経て（独）産業技術研究所情報セキュリティ研究センター主任研究員．日本クラウドセキュリティアライアンス ボードメンバ．情報理工学博士（東京大学）．