2004－ＤＰＳ－117　(32)
2004－CSEC－　24　(32)
2004／3／5

# 安全なジョブの遠隔実行を可能にする Trusted Platform on demand

丸山 宏*†　宗藤 誠治†　吉濱 佐知子†　Tim Ebringer†
*アイビーエム ビジネスコンサルティング サービス株式会社 東京都千代田区丸の内 2-4-1
†日本アイ・ビー・エム株式会社 東京基礎研究所 神奈川県大和市下鶴間 1623-14

**概要.** Trusted Computing Group (TCG)の技術を用いると、コンピュータプラットフォームのソフトウェア完全性を遠隔地から検証することができる。この機能と、強制アクセス制御を持つオペレーティングシステムを組み合わせることによって、セキュアなジョブの投入が可能なプラットフォーム Trusted Platform on demand (TPod) を構築した。その設計と実装について述べる。

# TPod - Trusted Platform on demand

Hiroshi Maruyama*†　Seiji Munetoh†　Sachiko Yoshihama†　Tim Ebringer†
* IBM Business Consulting Services, 2-4-1 Marunouchi, Chiyoda-ku, Tokyo, JAPAN
† IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato-shi, Kanagawa-ken, JAPAN

**Abstract.** In service-centric, “on demand” computing, establishing stronger trust on networked platforms is a key requirement because these remote platforms are often owned and managed by separate entities. The research described in this paper is an architecture and implementation called Trusted Platform on Demand (TPod), which increases the trustworthiness of networked platforms by combining dedicated security hardware, a secure operating system kernel and an open security protocol, to provide a secure software platform that may host a diverse range of distributed applications. Especially significant is the fact that the applications are better protected even if there are vulnerabilities in the application software or in the system software, or the system administrator is not completely trustworthy.

## 1. はじめに

複数のホストをまたがって実行される分散アプリケーションにおいては、個々のプラットフォームが同様に信頼できるものでなければならない。これらの構成要素が異なる企業によって管理されることの多い「オンデマンド」の世界では、それぞれ異なる管理ポリシーがアプリケーションの要件を満たすか検証する必要がある。さらに、それぞれのホストは、異なるアプリケーションを実行しているかもしれず、それら他のアプリケーションからの保護も重要な要件となる。

セキュリティ上の観点からは、我々は特にソフトウェアの脆弱性に基づく脅威を重視している。プラットフォームに脆弱性があれば、本来守られるべき管理ポリシーが実際には守られていないことになるからである。

この論文では、以上のような問題を解決する試みの一つとして Trusted Platform on demand (TPod)と呼ぶアーキテクチャを提案する。このアーキテクチャは、一般的なものであり、様々な個別技術で実装可能だが、我々は特に Trusted Computing Group (TCG), SELinux, OSGi 及び WS-Security を組み合わせた実装

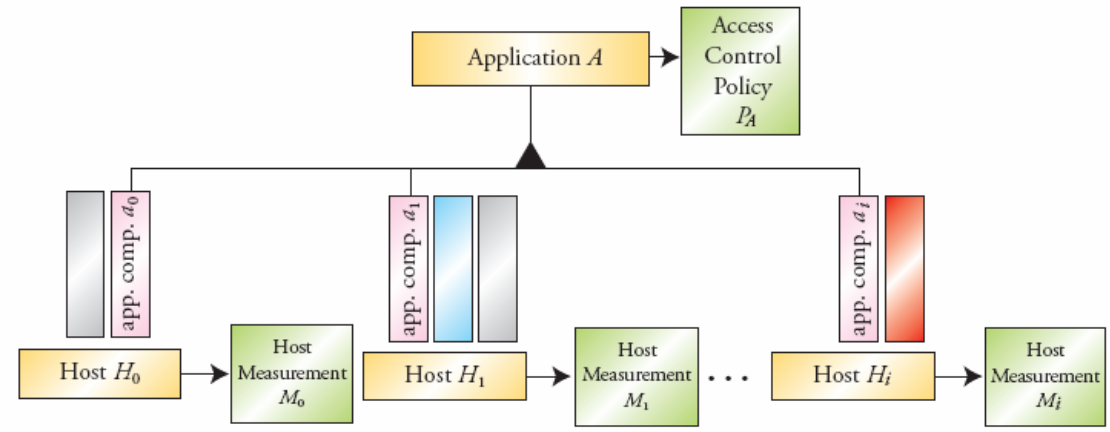


**図 1 TPod アプリケーションモデル**

を行ったのでそれもあわせて報告する。
次の節では我々が考える脅威モデルとそこから導かれるセキュリティ要件について議論する。3 節ではアーキテクチャの全体像を示し、4-7節ではアーキテクチャを構成する要素についてそれぞれ述べる。

## 2. 脅威モデルと要件

TPodが想定する分散アプリケーションモデルを図1に示す。アプリケーション *A* は異なるホスト H0, H1, …, Hi 上で実行されるコンポーネント*{a0, a1, …, ai}*から構成される。アプリケーション A はそのアプリケーション上のセキュリティ要件として、アクセス制御ポリシ PA を持つ。一方、ホスト H0, H1, …, Hi はそれぞれ異なる管理ドメインにあり、それぞれ異なるポリシーによって管理されている。
それぞれのホストHjはその詳細なハードウェア・ソフトウェア構成を示す測定値MHjを持つ。この測定値は外部から見て信頼できるものでなければならない。ホストHjの測定値は、Hjが実装しているポリシーを示している。Hjの上で動いているOSやその構成に脆弱性がある場合もあるので、これは本来Hjが意図しているポリシーは異なることがあることに注意しなければならない。
従って、アプリケーションのオーナから見れば、以下のようなセキュリティ要件を満たす必要がある。
**(a1)** Hj の公開された(意図された)ポリシーが、アプリケーションのポリシーを満たすことを検証できること
**(a2)** OS の脆弱性の判明などにより Hj の実装されているポリシーと意図されたポリシーの間に乖離があった場合、それを検出できること
さらに、
**(b1)** ホストのポリシーが、複数のアプリケーション間の相互干渉を防ぐこと
**(b2)** 異なる実装を持つプラットフォームに対して共通なプロトコルによって上記(a1), (a2)の機能が実施できること

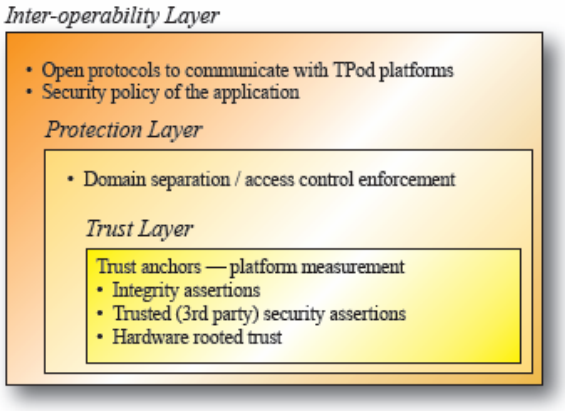


**図 2 TPod アーキテクチャ**

## 3. TPod アーキテクチャ

このような要件を満たすために、TPod は図 2 のように 3 つのレイヤからなる。最下層のレイヤは、Trust レイヤと呼び、ホストの測定とその検証に責任を持つ。中間層は保護レイヤと呼び、複数のアプリケーション間が相互干渉しないよう、ドメインの分離に責任を持つ。最上層は相互運用レイヤであり、オープンスタンダードに基づく TPod プラットフォームへのプロ

トコルを定義する。

## 4. Trust レイヤ

Trust レイヤの目的は、プラットフォームの完全性の検証を行うことである。プラットフォームの完全性が検証できれば、そのことから、例えば「この構成には現在知られているセキュリティ脆弱性がない」ということを確認することができる。

### 4.1. 完全性の測定

完全性の検証は信頼できる測定値から得られる。このためには、プラットフォーム上で信頼できる測定エージェントが存在しなければならない。

測定エージェントには、トラストモデルに依存して様々な形態が考えられる。

1. 信頼できる OS やミドルウェアなどのソフトウェアによる計測エージェント。
2. 専用ハードウェアによる計測エージェント。
3. 上記の組み合わせ

さらに計測エージェントは、計測値に基づいてプラットフォームの属性（例えば「この構成には現在知られているセキュリティ脆弱性がない」）を推論し、計測値自身に代わって推論された属性を報告することもできる。

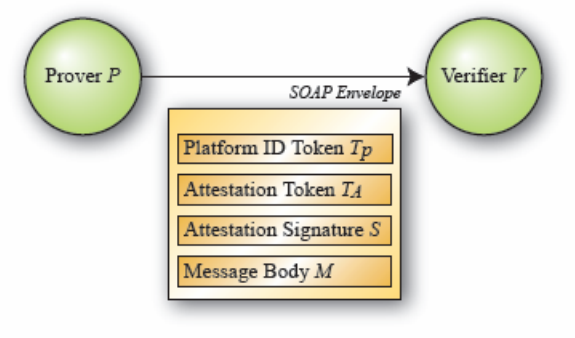


**図 3 送信者の完全性検証付きメッセージ**

### 4.2. Attestation プロトコル

測定された計測値あるいはそこから得られた属性を報告するためのプロトコルを Attestation プロトコルと呼ぶ。Attestation は、メッセージ本体 M に送信者の完全性測定値を付随させた形で、測定エージェントがデジタル署名 S を行うことによって行われる。この様子を図 3 に示す。Attestation トークン TA はこの測定値とそれが表す属性の結びつきを保証する。プラットフォーム ID トークン TP はプラットフォームの鍵に対する証明書である。このメッセージを受け取った検証者は、デジタル署名 S を調べることによって、メッセージ M と計測値とプラットフォームの身元の結びつきを知ることができる。さらに、この計測値がアプリケーションの要求するポリシーを満たすかどうかを判断する。これにはいくつかの戦略が考えられる。

1. 静的な検証。検証者はポリシーを満たす構成の計測値を自身のデータベースに予め保持している。検証者は報告された計測値をこのデータベースに保持された値と比べることによって、計測値がポリシーを満たすかどうかを知る。
2. 動的な検証。メッセージの送信者は信頼できる計測値と共に、システム構成の詳細を送る。検証者は(1)計測値が確かにそのシステム構成を示していること、(2)そのシステム構成が要求するポリシーを満たしていること、の 2 点をチェックする。
3. 第三者による保証。この場合、Attestation トークンは、計測値とその計測値が実現するポリシーに対して信頼する第三者機関が署名したものとなる。検証者は、計測値が Attestation トークンと一致し、また Attestation トークンに含まれるポリシーが要求するポリシーを満たし、かつ第三者による署名が信頼できるものであることを検証する。

### 4.3. Trust インフラストラクチャ

一般的には TPod プラットフォームの信頼のためには、第三者機関による保証が必要である。このために我々は図 4 に示すインフラストラクチャを仮定する。Identity CA はプラットフォームの鍵に対して証明書を発行する。Validation Entity は、検証の第 3 のシナリオ（第三者による保証）を実現するために、計測値とポリシーを結びつけるトークンを発行する。

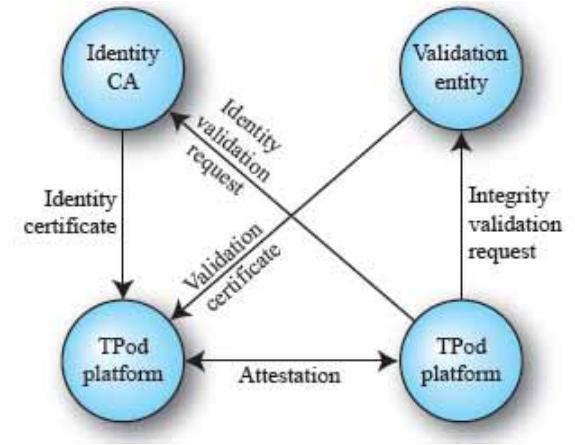


図 4 Trust インフラストラクチャ

## 5. TCG に基づく Trust プロバイダ

Trusted Computing Group (TCG)は TPod の Trust レイヤを実現可能な一つのテクノロジである。この節では、TCG を用いて Trust レイヤを実現した実装の詳細を示す。

### 5.1. Trusted Bootstrap

TCG による計測では、２つのハードウェアコンポーネントへの信頼を前提とする。一つは Trusted Platform Module (TPM)と呼ばれるセキュリティ・コプロセッサであり、もう一つは Core Root of Trust of Measurement (CRTM)と呼ばれる、BIOS の一部である。

システムが起動する際に、これらの２つのコンポーネントから、次々と完全性の計測値が計算される。この様子を図 5 に示す。

CRTM は、システムリセットの後に最初にコントロールを受け取るプログラムであり、通常書き換え不可能な 64KB の ROM である。CRTM は BIOS の残りの部分の完全性を計測し（SHA1 ハッシュ値を取ることによって計測）、その計測値を TPM に報告する。TPM はその値を Platform Configuration Register (PCR)と呼ばれるレジスタに、セキュアに格納する。

BIOS は次に OS のブートローダの完全性を検証し、その結果を TPM に報告する。その後、コントロールを計測済みのブートローダに移す。ブートローダは、OS をロードする前に、OS カーネルのイメージを計測し、同様に TPM に報告する。その後、OS をロードし、OS にコントロールを移す。

この時点で、TPM の PCR レジスタにはこの特定の BIOS、ローダ、OS の組み合わせに特

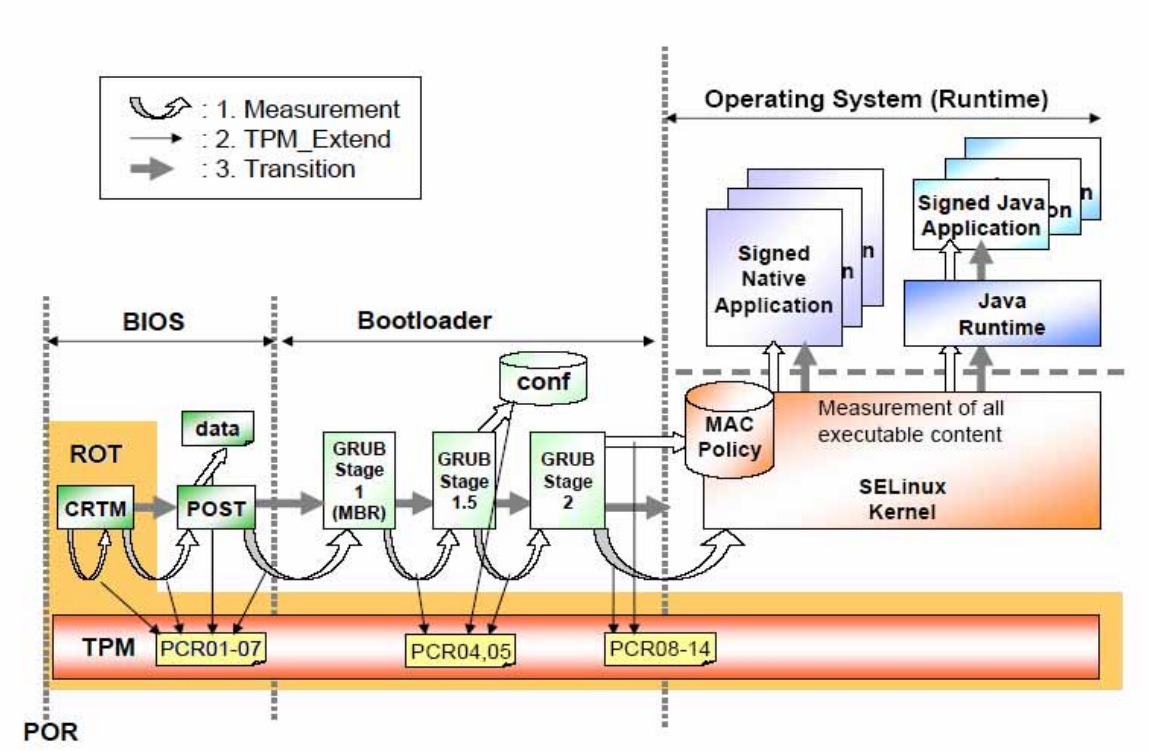


図 5 Trusted Bootstrap

有の測定値が入っている。この値を知れば現在の構成を知ることができる。
我々の TPod の実装では、ハードウェアとして TCG 1.1b 準拠の TPM を持つ ThinkPad R32、OS ローダとして GRUB、OS として SELinux を用いた。ThinkPad の BIOS には上記の計測の仕組みが含まれている。ブートローダは、GRUB を変更し、ロードする OS の測定を行うようにした。

## 6. 保護レイヤ

TPod の保護レイヤは２つの目的を持つ。一つは、計測されたプラットフォームの完全性が、計測後に失われないように攻撃や事故から保護することであり、もう一つは、複数のアプリケーションが互いに干渉しないよう、保護することである。

### 6.1. ドメイン

TPod プラットフォーム上のアプリケーションはドメインの中で動く。ドメイン同士の間はプラットフォームの強制アクセス制御(MAC)の仕組みによって隔離される。また、ドメインが保有するデータへのアクセスは、ドメインが規定する任意アクセス制御（DAC）によって制限される。この DAC ポリシーは、アプリケーションのポリシーを反映する。
我々の実装では、MAC ポリシーは SELinux と OSGi で実現される。SELinux はブートストラップ後、すなわち計測値が TPM に格納された後にプラットフォームの完全性を保護するのに必須である。殊に、カーネルモジュールを追加する/sbin/insmod コマンドを禁止したり、メモリへの直接のアクセスである /dev/kmem へのアクセスを禁止したりしておく必要がある。また、SELinux の強制アクセスポリシーを適切に設定することによって、ホストの管理者からもドメインデータへアクセスを制限することができる。
一方、Java アプリケーションに対する強制アクセス制御は、OSGi のフレームワークを用いた。OSGi では、アプリケーションは「サービスバンドル」という単位で提供され、それぞれのバンドルの間は独立である。
我々が想定する使い方では、プラットフォームはブートストラップ後、SELinux のポリシーで許される単一の Java 仮想マシンが、Security Manager を enable した状態で立ち上がる。その後 Java 2 の標準的なセキュリティ・モデルによってアクセス制御が行われる。

### 6.2. ドメイン・インスタンスの生成

分散アプリケーション A がホスト H 上でそのコンポーネント aj を動かそうとする場合、H の上にドメインのインスタンスを作らなければならない。この際、アプリケーション A は、ドメイン･インスタンス生成プロトコルを実行する。このドメイン・インスタンス生成プロトコルは以下のステップからなる。

1. 相互認証

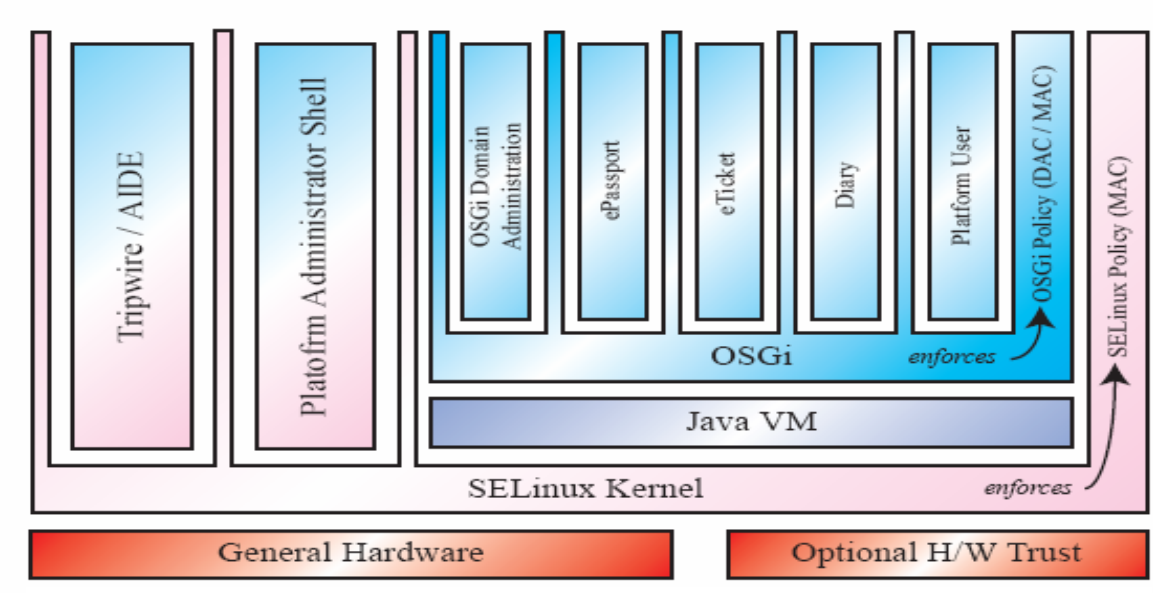


**図 6 TPod ドメインモデル**

2. ホスト Hj の Attestation、Hj のポリシーがアプリケーション A のポリシーを満たすことを検証
3. ドメイン・インスタンスの生成
4. アプリケーションコンポーネント aj の実行
5. ドメイン・インスタンスの消去

これらのステップを経ることによって、アプリケーション A は Hj がポリシー要件を満たすときに限り、ドメインを生成し aj を実行することになる。

### 6.3. 永続性のある記憶

アプリケーションコンポーネント aj はアプリケーションのデータをホスト Hj 上に格納しておきたいかもしれない。しかしながら、永続性のあるデータは、ホストがオフラインの時、あるいは別の OS が立ち上がっている時に攻撃者によってアクセスされるかもしれない。これを避けるために、TPod プラットフォームのファイルシステムは、ドメイン毎の鍵によって暗号化される。この鍵は、ドメイン・インスタンスが生成される際、すなわちアプリケーションが Hj の完全性とポリシーを確認した上で Hj に渡される。コンポーネント aj の実行が終了すると、ドメイン・インスタンスは消滅し、ドメイン・インスタンスが使用していたメモリの内容は消去される。同時に、ドメイン鍵もメモリから消去される。これにより、ドメインデータの保護が可能になる。

### 6.4. TCB の更新

TCB に脆弱性が見つかった場合、TCB を更新しなければならない。TCB の更新の際には、すべてのドメイン・インスタンスを消去し、TCB への修正を行ったうえで、システムがリスタートされる。これによって、PCR には、新しいシステム構成に基づく測定値が格納され、アプリケーションはこの値に基づいてポリシーが守られるかどうかの判断を行う。

## 7. 相互運用性レイヤ

我々の現在の実装は Linux ベースの実装であるが、TPod は一般的なアーキテクチャであり様々な実現がありうると考えている。そのためには、TPod プラットフォームに対しては共通のプロトコルで通信できなければならない。我々は、Attestation プロトコルを含むドメイン・インスタンス生成プロトコルを WS-Security を用いて SOAP 上に定義しつつある。また、インフラストラクチャは WS-Trust のプロファイルを定義することによって、相互運用性を保ちつつ完全性検証のための機能を提供しようとしている。

## 8. 終わりに

遠隔地にあり、異なる管理ドメイン上のホストにおいて安全に分散アプリケーションのコンポーネントを実行できる枠組み TPod を提案した。また、そのプロトタイプを、TCG, SELinux, OSGi,及び WS-Security という標準を組み合わせることで実現した。今後はこのプロトタイプを様々な場面でテストする予定である。

## 参考文献


[1] OASIS TC draft. Web services security core. http://www.oasis-open.org/committees/wss/, December 2002.

[2] Trusted Computing Group. TCG PC specific implementation specification v1.1. https://www.trustedcomputinggroup.org/downloads/TCG_PCSpecificSpecification_v1_1.pdf, September 2001.

[3] Trusted Computing Group. TCG main specification v.1.1b. https://www.trustedcomputinggroup.org/downloads/tcg_spec_1_1b.zip, February 2002.

[4] Andrew Huang. Keeping secrets in hardware: The microsoft xbox case study. In Cryptographic Hardware and Embedded Systems, volume 2523 of Lecture Notes in Computer Science, pages 213–227. Springer-Verlag, Heidelberg, 2002.

[5] Greg Kroah-Hartman. Signed kernel modules. Linux Journal, 117:48–53, Jan 2004. Available at http://www.linuxjournal.com/article.php?sid=7130.

[6] Open grid services architecture. http://www.globus.org/ogsa/.

[7] Open services gateway initiative (OSGi). http://www.osgi.org/.

[8] L. Van Doorn, G. Ballintijn, and W. A. Arbaugh. Signed executables for linux. Technical Report UMD CS-TR-4259, University of Maryland, June 2001. http://www.ece.cmu.edu/~leendert/publications/