社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－DPS－138 (1)
2009－CSEC－ 44 (1)
2009／3／5

カーネル保護機能を利用したファイル完全性リモート検証機構

竹森 敬祐† 磯原 隆将† 三宅 優†

†KDDI 研究所 〒356-8502 埼玉県ふじみ野市大原 2-1-15

あらまし データセンタで運用されるサーバの管理者は，重要なファイルの完全性をリモート検証することで侵入の有無を検知する．ここで，Rootkit をカーネルに埋め込んで応答を偽る攻撃，ファイル状態を測定するアプリケーションを改竄して応答を偽る攻撃，測定結果を改竄して応答を偽る攻撃などがあり，リモート検証の信頼性を担保する必要がある．昨今，Trusted Platform Module（TPM）を用いて，カーネルとアプリケーションの完全性をリモート検証する仕組が提案されている．しかし，TPM は起動時にのみブートローダ⇒カーネル⇒アプリケーションへと信頼の連鎖を繋ぐのみであり，稼動し続けるサーバの HDD ファイルの検証は行えない．そこで本研究では，起動時のみならず稼動中のサーバの HDD ファイルをリモート検証する機構として，ファイル測定アプリケーションが完全であることを，完全性が担保されたカーネル内で検証して起動させることで，カーネル⇒ファイル測定アプリケーション⇒HDD ファイルへと信頼の連鎖を繋ぐ仕組を提案する．また，TPM で管理される秘密鍵で測定結果に電子署名を施すことで，測定結果の改竄の有無をリモートで検証できる仕組を提案する．

キーワード リモート検証，カーネルとファイルの完全性，Linux Security Module，信頼の連鎖

Remote Attestation for HDD Files with Lifetime Kernel Protection

Keisuke TAKEMORI†, Takamasa ISOHARA† and Yutaka MIYAKE†

†KDDI R&D Laboratories Inc. 2-1-15 Ohara, Fujimino-city, Saitama, 356-8502 Japan

Abstract A remote attestation technique that checks integrity of system-files on a data center server is important for intrusion detection. When the server is infected by a rootkit, the kernel replies faked responses. Also, when a file measurement application or its result is manipulated, a measurement response is not reliable. A trusted platform module (TPM) is proposed for applying to the remote attestation. The TPM can make a trust chain from a boot loader via a kernel to user applications when the server starts up. Because the data center server is rarely stopped, the TPM is not suitable for the remote attestation. In this research, we propose a trusted file attestation scheme that makes a trust chain from kernel via user applications to HDD files on a running server to implement a trusted file measurement application in a trusted kernel. Also, we propose a remote verification scheme that attaches a signature at the TPM to the attestation result.

Keywords Remote Attestation, Kernel and File Integrity, Linux Security Module, Trust Chain

1. はじめに

Web サーバへの侵入によるホームページの改竄事件が跡を絶たない．このため，定期的に重要なファイルのハッシュ値を算出することで状態を測定し，完全性を検証する必要がある．ここで，データセンタなどの立ち入りが制限されている場所でサーバが運用されている場合には，リモートからファイル状態を測定することになる．しかし，サーバが侵害されている場合には，Rootkit をカーネルに埋め込んで応答を偽る攻撃，ファイル測定アプリケーションのコードを改竄して応答を偽る攻撃，測定結果を改竄して応答を偽る攻撃などが想定され，リモート側で完全な測定結果を受け取れる保証がない．

昨今，多くのサーバには TPM[1]とよばれる耐タンパ性を有するチップが搭載されており，これを使ったカーネルとアプリケーションの完全性をリモート検証する仕組が提案されている[2]．しかし，TPM は起動時にのみ，ブートローダ⇒カーネル⇒アプリケーションへと信頼の連鎖を

繋ぐのみであり，稼動中の公開サーバの検証には不向きである．

そこで本研究では，稼動し続けるサーバのHDD ファイルの完全性をリモート検証する機構として，完全なファイル測定アプリケーションの起動制御を，完全性が担保されたカーネル内に組み込むことで，HDD 上のシステムファイルの完全性を検証する仕組について提案する．稼動中のカーネルの完全性は，SecVisor [3]と呼ばれるユーザメモリ空間からカーネルメモリ空間へのアクセスを制限する技術によって担保する．そして，セキュア OS を実装するためのフレームワークである Linux Security Module（LSM）が提供するカーネル内処理のチェックポイントに，アプリケーションに付与された電子署名を検証して起動させる DigSig[4]と呼ばれる手法を組み込み，これをファイル測定アプリケーションの起動制御に適用する．また，ファイル測定アプリケーションが，HDD 上のファイル状態を測定して，その結果に対して安全なカーネル内で電子署名を施すことで，測定結果の完全性をリモート側で検証できる仕組も提案する．これにより，TPM では実現できなかった稼動中のサーバに対する，カーネル⇒ファイル測定アプリケーション⇒HDD ファイルへの信頼の連鎖を繋ぐことができ，リモートの管理者が信頼できる測定結果を得ることができるようになる．

以下 2 章において関連する技術を紹介する．3 章で提案するリモート検証機構の概要を説明し，4 章で完全なカーネルから完全なアプリケーションを起動する仕組と，ファイル測定アプリケーションの結果に対して TPM を使って電子署名を施す仕組を提案する．5 章で，実装したシステムの処理負荷に関する評価を行い，そして最後に 6 章で纏める．

2. 関連研究

2.1. TPM：起動時の完全性の検証

TPM[1]は，外部からアクセスできない安全な領域を持った計算能力を有する耐タンパチップであり，その安全な領域に，ハッシュ計算，鍵ペアの生成，電子署名などの機能，計算結果を格納するメモリなどを持っている．TPM は，カーネルから独立した処理を行えるため，カーネルが改竄された場合でも，その影響を受けない．

TPM はホストが起動するタイミングでロードされるモジュールのハッシュ値を保存する機能を持ち，BIOS⇒ブートローダ⇒カーネル⇒アプリケーケーションの状態を連鎖的なハッシュ値として管理できる[2]．そして，外部からの要求に従い，電子署名を施した連鎖的なハッシュ値を返信する．これにより，起動時のホストの完全性をリモートの管理者が把握することができるようになる．TPM を起点とした信頼の連鎖を実現している様子を図 1 に示す．

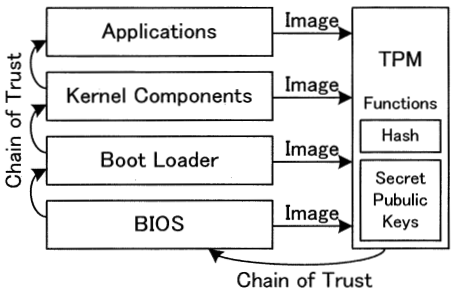

図 1. TPM を用いた起動時の完全性の測定

2.2. SecVisor：起動時および稼動中カーネルの完全性の担保

SecVisor[3]は，ブートローダの代わりにシステムファイルを測定して完全なカーネルをロードする役割と，ユーザアプリケーションからカーネルメモリへのアクセス権限を制御することで，起動中のカーネルメモリに対する不正な書き込みを禁止する仕組を持つ．

図 2 に，AMD 社の Secure Virtual Machine（SVM）を搭載した CPU が持つ SKINIT[5]の機能を用いて，起動時に BIOS と SecVisor ブートローダのイメージを TPM に送り込み，TPM で管理される正しいハッシュ値と比較して，完全性を検証できた場合にのみ BIOS と SecVisor ブートローダに起動を許可する様子を示す．SecVisor ブートローダが起動すると，カーネルファイルのハッシュ値を算出し，SecVisor で管理される正しいハッシュ値リストと比較して，完全性を検証できた場合にのみカーネルをロードする．

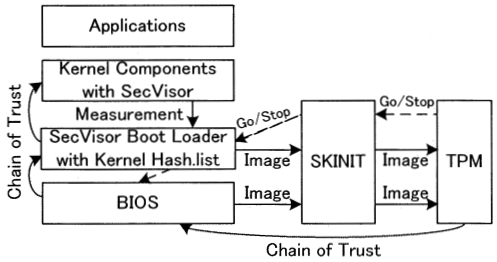

図 2. SecVisor によるシステムの測定と起動制御

図 3 に，SVM が持つ Nested Page Table（NPT）というメモリをハードウェア的に仮想化する技術を用いて，CPU がユーザモードならびにカーネルモードで動作しているときの，メモリへのアクセス権限を制御する様子を示す．カーネル本体は，カーネルコード領域にロードされており，この領域に対する書き込み権限を与えないことで稼動中のカーネルの改竄を防止できる．また，カーネルデータ領域に悪意のコードが置かれた場合にも，これを実行させないために，実行権限を外しておく．NPT を持たない CPU についても，ソフトウェア的にメモリ空間を仮想化する Shadow Page Table（SPT）という技術があり，これを用いて，図 3 のようなアクセス権限制御を行うことができる．また SecVisor は，Direct Memory Access（DMA）デバイスからカーネルメモリに対する直接的な書き込みを禁止する仕組として，Input/Output Memory Management Unit（IOMMU）を制御して，カーネルコード領域へのアクセスを禁止する仕組みも持つ．

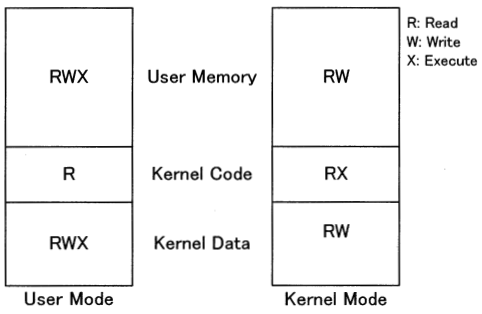

図 3. NPT を用いたユーザモードとカーネルモードのメモリアクセスの権限制御

2.3. DigSig：アプリケーションの署名者の認証と完全性の検証

DigSig[4]は，あらかじめ電子署名をアプリケーションに施しておき，起動時の処理をカーネル空間の LSM でフックして署名検証することで，信頼された完全なアプリケーションのみを起動させる制御を実現している．

図 4 に，LSM を用いてユーザアプリケーションに付与された電子書名の検証を行い，完全性が確認された場合にのみ起動を許可する DigSig の技術を紹介する．ユーザアプリケーションの起動を要求する sys_execve がコールされると，do_execve, search_binary_handler, load_elf_binary, do_mmap などのシステムコールがカーネル内で連鎖的に呼び出され，最終的には file_mmap という LSM のチェックポイントを経て，アプリケーションのバイナリイメージと電子署名を含むヘッダ情報がカーネルに読み込まれる．この file_mmap において，バイナリイメージのハッシュ値と，電子署名を復号化して取り出したハッシュ値が一致すれば，署名者の認証とバイナリファイルの完全性が担保され，ユーザメモリへのロード処理へと進む．

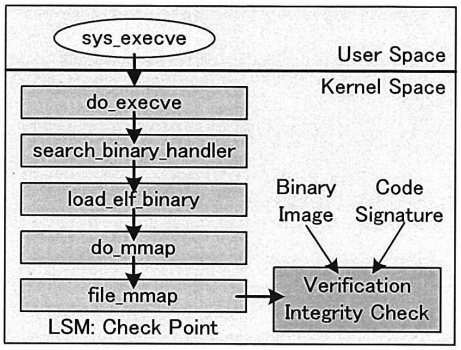

図 4. 電子署名を用いた起動アプリケーションの完全性の検証

2.4. リモート検証のための要件

稼動中サーバの HDD ファイルの完全性をリモート検証するためには，完全なカーネルから，完全なファイル測定アプリケーションを起動させて HDD ファイルのハッシュ値リストを作成し，このリストへ安全に電子署名を施した後に外部の検証者へ返信する必要がある．

3. HDD ファイルのリモート検証の概要

本研究では，稼動し続けるサーバの HDD ファイルの完全性を確実にリモート検証する手法を提案する．図 5 に提案モデルの構成を示す．

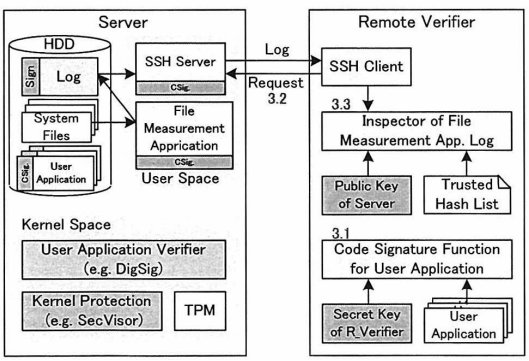

図 5. 稼動サーバの HDD ファイルのリモート検証

3.1. 電子署名の付与

図 5 右側のリモート検証者は，サーバのユーザメモリで稼動する全てのアプリケーションに対して，自身の秘密鍵で電子署名を施す．電子署名は，信頼されたアプリケーションの印となり，署名者の認証とコードの完全性の検証に用いられる．

図 5 左側のサーバ上では，電子署名の検証を経た完全なアプリケーションしか起動しない環境を構築することとし，あらかじめリモート検証者がサーバに電子署名の付いたアプリケーションをインストールしておく．

3.2. リモート検証の要求

リモート検証者は，SSH サービスを利用して，サーバ HDD 上の重要なファイルのハッシュ値リストを要求する．サーバ上の SSH サービスは，3.1 節で電子署名の付された完全性の担保されたアプリケーションである．SSH サービスを通じて，同じく完全なファイル測定アプリケーションを起動し，HDD 上の重要なファイルに関するハッシュ値リストを作成する．そして，サーバの TPM が管理する秘密鍵で電子署名を施す．

この測定は改竄検知を目的としており，対象は変化してはならないアプリケーションのバイナリコードやサーバの設定ファイルである．

3.3. 測定結果の検証

3.2 節の測定で得られた重要なファイルのハッシュ値リストは，信頼される SSH サービスを通じて，リモート検証者に返信される．リモート検証者は，サーバの公開鍵を利用して返信された結果に付随する電子署名を検証する．

リモート検証者は，測定対象となったファイルの正しいハッシュ値リストを保有しており，返信された測定結果と突き合わせて，改竄がないことを確認する．

4. 完全なカーネルから完全なアプリケーションを起動させる制御

ここでは，稼動中のカーネルの完全性を担保する技術と，認証されたアプリケーションのみを起動する技術を組み合わせることで，起動時のみならず稼動中においても，ブートローダ⇒カーネル⇒アプリケーションへと信頼の連鎖を繋ぐ仕組について説明する．

4.1. 前提条件

ファイル測定アプリケーションの入力インタフェースは，ファイルを読み込む機能しかない．この機能が安全に実装されていれば，攻撃者からのバッファオーバフロー攻撃を受けることはない．よって，ユーザメモリに展開されたファイル測定アプリケーションは，攻撃を受けることなく，処理を完了できるものとする．

4.2. BIOS⇒ブートローダ⇒カーネルへの信頼の連鎖

図 2 で説明したように，起動時に SKINIT と TPM が連携して，BIOS と SecVisor ブートローダの完全性を担保している．

次に SecVisor ブートローダは，カーネルファイルの測定を行い，自身が管理しているカーネルファイルの正しいハッシュ値リストと比較して，問題がなければカーネルをメモリにロードする．これにより，BIOS⇒ブートローダ⇒カーネルへの信頼の連鎖が繋がる．

ひとたびカーネルがメモリ上にロードされると，SecVisor の SPT による保護機構が働き，完全なカーネルが稼動し続ける．

4.3. カーネル⇒アプリケーションへの信頼の連鎖

ここでは，稼動中のサーバにおいて完全なカーネル上で，アプリケーションの電子署名を検証して，署名者の認証を行い，完全性が担保されたアプリケーションのみを起動させることで，カーネル⇒ユーザアプリケーションへの信頼の連鎖を作り出す手法を説明する．電子署名の検証は，file_mmap()と呼ばれるファイル操作に関わる LSM のチェックポイントに実装した．この様子を図 6 に示す．

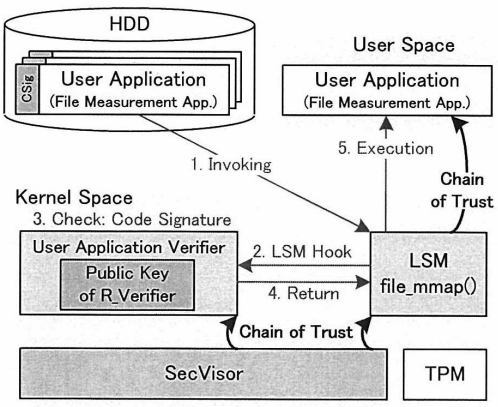

図 6. 完全なカーネルから信頼される完全なアプリケーションを起動させる仕組

1) ユーザアプリケーションの起動要求が入る．
2) LSM の file_mmap()のチェックポイントで起動要求をフックする．
3) リモート検証者の公開鍵を用いて，ユーザアプリケーションの電子署名を検証する．検証は，公開鍵で復号されたハッシュ値と，ユーザアプリケーションのバイナリコードのハッシュ値を比較して，一致すれば署名者の認証とコードの完全性が担保される．
4) 検証が正しく行われた場合には，本来の起動処理へと戻る．検証の結果，改竄が検知された場合には，起動処理を中止する．
5) 正しく検証された場合には，アプリケーションをユーザメモリへロードして起動する．

4.4. ユーザアプリケーション⇒HDD ファイルへの信頼の連鎖

4.3 節で起動されるアプリケーションの一つは，HDD 上のファイルのハッシュ値を算出してリストを作成する測定アプリケーションとする．ここで，カーネルから直接 HDD ファイルの状態を測定する手法も考えられるが，測定対象ファイルの設定や測定制御を柔軟に行うために，専用のアプリケーションを設けることとした．

図 7 を用いて，ファイル測定アプリケーションが HDD ファイルの測定を行い，その結果をリストとしてファイル出力する前に，安全に電子署名を施す手順について説明する．

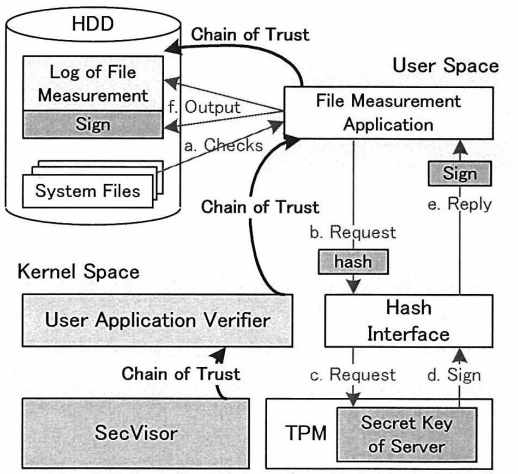

図 7. TPM を用いた測定結果に対する電子署名

a) ファイル測定アプリケーションは，指定された重要なファイルを読み込み，そのハッシュ値を算出する．結果は，ハッシュ値リストとして自身が確保したユーザメモリ上で保持しておく．また，ハッシュ値リストのハッシュ値も算出する．
b) ハッシュ値を受け取るキャラクタベースのインタフェース(IF)をカーネルに設けておく．ファイル測定アプリケーションは，ハッシュ値リストのハッシュ値を，この IF を通じてカーネルに引き渡す．
c) カーネル上の IF は，ハッシュ値を受け取り，TPM に中継する．
d) TPM は，安全な領域で管理しているサーバの秘密鍵で，受け取ったハッシュ値を暗号化する．これは，ファイル測定アプリケーションのログファイルに対する電子署名となる．
e) IF は，電子署名をファイル測定アプリケーションへと返信する．
f) ファイル測定アプリケーションは，測定結果であるハッシュ値リストと，その電子署名を HDD に書き出す．

書き出された測定結果と電子署名は，3 章で説明したリモート検証者に送付される．

5. 評価

ここでは，カーネル保護機構とアプリケーション認証機構の両者を実装したサーバ上で，リモート検証の際の処理負荷について評価する．

5.1. 実装システム

我々は，提案システムを以下のサーバに実装した．

- HP Compaq dc5750 Microtower
- 2.2 GHz AMD Athlon64(dualcore CPU) SVM hardware virtualization support
- 2GByte RAM
- Linux 2.6.20.14

5.2. ユーザアプリケーションの署名検証時間

本システムは，認証された完全なアプリケーションのみが起動するサーバを想定している．そこで，この署名検証に要する処理時間を把握するために，“Hello”というメッセージをディスプレイに表示するアプリケーションを作成し，このアプリケーションが起動・終了するまでの処理時間を測定した．測定は，本アプリケーションを 10 万回繰り返し起動･終了させたときの，1 回目と最終回の時刻を Time コマンドで取得して，1 回あたりの処理時間を算出した．ここで処理時間とは，表 1 に示す CPU 処理時間のことで，プログラムの起動・終了に関わる一連の処理

と"Hello"を表示する処理の両者が含まれる．測定条件は，表 2 に示す(i)Linux カーネル，(ii)署名検証処理を実装した Linux カーネル，(iii)SecVisor 保護カーネル，(iv)署名検証処理を実装した SecVisor 保護カーネルの 4 パターンである．

評価結果を図 8 に示す．Linux2.6.20.14 カーネル上で 1 回あたりの署名検証処理に要する CPU 時間は，(ii)－(i)＝0.25msec であり，16%のオーバヘッドで済むことがわかる．これは，初めの 1 回目のみ公開鍵による認証と完全性検証を行い，2 回目以降は hash 計算による完全性の検証のみという効率化が図られている為である．次に，カーネル保護機構である SecVisor を適用したときの起動・表示・終了処理時間は，(iii)＝8.28msec で，何も施していないカーネルの処理時間に対して 428%のオーバヘッドとなった．これは"Hello"メッセージをディスプレイに表示する際のカーネル処理を SecVisor がフックしている負荷である．最後に，SecVisor の適用されたカーネル上での起動・署名検証・表示・終了処理時間は，(iv)=98.44msec で 6,179%のオーバヘッドであった．このオーバヘッドの多くは，保護されたカーネル上での LSM フックによる署名検証時間であり，(iv)－(iii)＝90.16msec となる．この 90msec の CPU 時間は，アプリケーションの完全性検証のための hash 処理を，カーネル内で行う際の SecVisor の介在が原因であり，プロセス起動時の特有のものである．尚，オーバヘッドの増加率は高いものの，時間的には，ユーザが感じるほどではないことに注意されたい．

表 1. 処理時間の説明

時間種別	説明
実時間	コマンドを実行してから終了するまでの時間.
CPU時間	プログラムがCPUを使用した時間. ディスクアクセスやDMA転送時間は含まれない. CPU時間＝ユーザCPU時間＋システムCPU時間
ユーザCPU時間	プログラムの処理について, ユーザ空間上でCPUを使用した時間.
システムCPU時間	プログラムの依頼でカーネルがCPUを使用した時間.

表 2. 評価パターン

条件	説明
(i)	Linux 2.6.20.14
(ii)	Linux 2.6.20.14 ＋ 署名検証処理
(iii)	Linux 2.6.20.14 ＋ SecVisor保護
(iv)	Linux 2.6.20.14 ＋ SecVisor保護 ＋ 署名検証処理

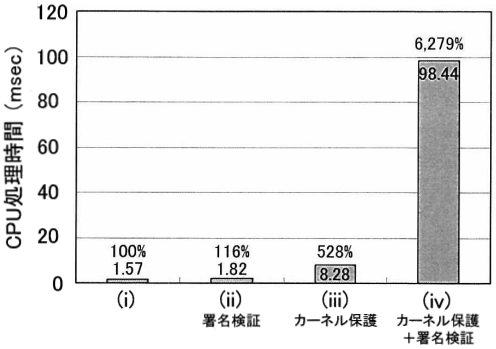

図 8. 署名検証処理に要するオーバヘッド

6. おわりに

本稿では，完全性が担保されたカーネル内に，完全なファイル測定アプリケーションを起動させる機能を組み込み，このファイル測定アプリケーションが HDD ファイルを測定することで，ブートローダ⇒カーネル⇒アプリケーション⇒HDD ファイルへと信頼の連鎖を繋ぐ方式を提案した．また，TPM で HDD ファイルの測定結果に電子署名を施すことで，リモートの管理者がサーバの HDD 状態を確実に把握できる仕組を提案した．これにより，起動時のみならず稼動し続けるサーバに対しても，HDD ファイルの完全性をリモート検証できるようになる．

Acknowledgement

This research collaborates with Adrian Perrig and Ning Qu at Carnegie Mellon University (CMU). We appreciate their active supports.

参考文献

[1] Trust Computing Group: https://www.trustedcomputinggroup.org/home

[2] 中村めぐみ，宗藤誠治，工藤道治，“プラットフォーム完全性をベースとした脆弱性検証システム”，CSS2007，pp.429-434，2007 年 10 月．

[3] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig, “SecVisor: A Tiny Hypervisor to Provide .Lifetime Kernel Code Integrity for Commodity OSes,” ACM, SOSP Symposium, October, 2007.

[4] A. Apvrille, M. Pourzandi, D. Gordon, and V. Roy, “Stop Malicious Code Execution at Kernel-Level,” Linux World, Vol.2, No.1, January, 2004.

[5] ITpro, ”仮想化技術”, http://itpro.nikkeibp.co.jp/members/NBY/ITARTICLE/20050627/163385/