日本ソフトウェア科学会第 27 回大会 (2010 年度) 講演論文集

Cell/B.E. のSPE上で動作する安全なOS監視システム

永田 卓也　光来 健一

近年、OS が改ざんされていないことを保証する必要性が高まっている。ソフトウェアは OS の上で動き、セキュリティソフトウェアも例外ではない。OS が攻撃を受け、改ざんされた場合、ソフトウェアが正しく動くことを保証することはできない。そこで、OS が正しく動作していることを監視すべきであるが、従来のシステム構成では OS 監視システムも改ざんされた OS 上で動いてしまう。

本研究では、Cell/B.E. の SPE が持つ Isolation モードを用いて、OS を動かす PPE から隔離された SPE 上で OS カーネルを安全に監視する SPE Observer を提案する。Isolation モードを用いることにより、SPE 上で動作する OS 監視システムの完全性と機密性を保証する。さらに、外部のセキュリティプロキシから OS 監視システムに定期的にハートビートを送ることで、OS 監視システムの可用性を向上させる。我々は PS3 上に SPE Observer を実装し、カーネルの改ざん検知および性能評価を行った。

1　はじめに

近年、ネットワークを経由して計算機が攻撃されることが増えている。各ユーザは AntiVirus 等のセキュリティソフトウェアを利用してそれらの攻撃に備えるのが一般的である。しかし、セキュリティソフトウェアは OS の機能を利用して監視を行っており、adore や knark、SucKIT 等のカーネルルートキットからの攻撃によって OS が改ざんされてしまった場合、セキュリティソフトウェアの実行結果を信頼することはできなくなる。

セキュリティソフトウェアの信頼性を向上させるためには OS が正しく動作していることを保証すべきである。TPM 等のハードウェアを用いることで、OS の完全性を保証することができる。しかし完全性チェックはブート時のみに行われるため、システム実行中の OS が改ざんされていないことを保証するのは困難である。カーネルルートキットはシステムの実行中に感染するため、システムが動作している間も OS の完全性を保証する必要がある。

本研究では、Cell/B.E. のもつ SPE Isolation モードに着目し、SPE を利用して OS を安全に監視するシステム SPE Observer を提案する。SPE Isolation モードを用いることにより、SPE 上で動く OS 監視システムの完全性および機密性を保証する。さらに、外部のセキュリティプロキシから OS 監視システムに定期的にハートビートを送ることで、OS 監視システムの可用性を向上させる。

我々は PS3 Linux 上に SPE Observer を実装し、SPE 上でカーネルメモリをチェックする監視システムを開発した。この監視システムを用いて、カーネルの改ざん検知と性能評価を行った。実験により、カーネルの改ざんを検知することができ、OS 監視システムの実行が他のコアのメモリ転送性能に影響を及ぼさず、SPE Observer 用にコアを占有した場合の性能低下は、コア 1 つ分で済むことが分かった。

以下、2 章で従来の OS 監視システムとその問題点について述べる。3 章で SPE Observer を提案し、4 章でその実装について述べる。5 章で SPE Observer の性能について述べる。6 章で関連研究に触れ、7 章

An OS Monitoring System Running on an SPE in Cell/B.E.

Takuya Nagata Kenichi Kourai, 九州工業大学, Kyushu Institute of Technology.

で本稿をまとめる。

2 従来の OS 監視システム

一般的なシステム構成では OS の上でセキュリティソフトウェアが動いており、OS に対してシステムコールを発行しながらセキュリティチェックを行っている。攻撃者が OS を改ざんした場合、システムコールの処理内容を変えられてしまう恐れがある。そのため、改ざんされた OS 上で動くセキュリティソフトウェアは正常な動作をしているとは言えない。

例えば、ウィルススキャンソフトウェアはファイルを開き、その内容をチェックし、結果をユーザに通知するという処理を行っている。ファイルを開く動作や結果の出力は OS へのシステムコールを使って実行されている。OS が改ざんされた場合には、攻撃に使うファイルが存在しないように見えたり、感染したファイルの内容が正常に見えたりするように騙される可能性がある。また、異常を検知してもユーザ側に通知できないという恐れもある。

そこで、OS が改ざんされていないことをチェックする必要があるが、OS 監視システムが OS の上で動いている場合、OS が改ざんされていないことを保証するのは難しい。OS 内部に OS 監視システムを埋め込むことも可能だが、改ざんされた OS と同一階層で動いている OS 監視システムが正しく動くという保証はない。

OS より下層にある TPM [8] 等のハードウェアを用いることで、OS が改ざんされていないかどうかをチェックすることができる。ハードウェアから chain of trust が繋がっていれば、ハードウェアそのものは改ざんすることができないため、OS の完全性を保証することができる。しかし、TPM が OS の整合性をチェックするのは起動時のみであるため、システムを動かしている間に OS が改ざんされたことを検知することはできない。

仮想マシンを用いて OS より下層で監視を行う手法 [3] も存在する。仮想マシン技術は、ゲスト OS の下に仮想マシンモニタ（VMM）と呼ばれるソフトウェアを走らせ、ゲスト OS からハードウェアへのアクセスを仮想化する技術である。この技術を用いると、ゲスト OS よりも下層で動く VMM がゲスト OS の監視を行うことで、ゲスト OS が改ざんされていないことを保証することができる。しかし、VMM はソフトウェアであるため、複数のバグが存在する可能性がある。例えば、バグの中にはゲスト OS から VMM に対して攻撃を行えるバグも報告されている [1] [2]。VMM が改ざんされてしまえば、その中で動く監視システムも正常に動く保証がなくなる。さらに、VMM が仮想化を行うことによるオーバーヘッドも存在する。

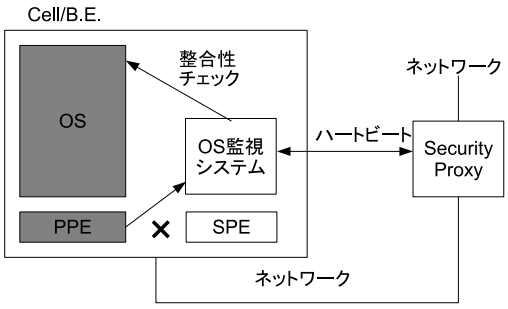

図 1 SPE Observer のシステム構成

3 SPE Observer

本研究では、Cell/B.E. の持つ SPE Isolation モードに着目し、OS が動いている PPE から物理的に隔離された SPE 上で OS の監視を行う SPE Observer を提案する。SPE Isolation モードを用いることにより OS 監視システムの完全性と機密性が保証される。また、外部のセキュリティプロキシから OS 監視システムに対してハートビートを送ることで OS 監視システムの可用性を向上させる。図 1 に SPE Observer の全体図を示す。

3.1 Cell/B.E.

Cell/B.E. [7] は OS や通常のプロセスを動かす PPE という制御コアと、演算処理に特化した SPE にという演算コアから構成されるヘテロジニアス・マルチコアプロセッサである。各コアは図 2 のように EIB と呼ばれるリングバスでつながれており、EIB を通じてメインメモリにアクセスを行う。SPE はロー

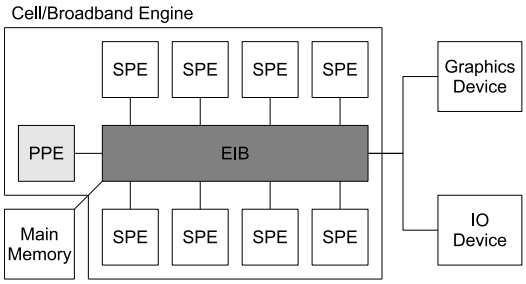

図 2 Cell/B.E. のアーキテクチャ

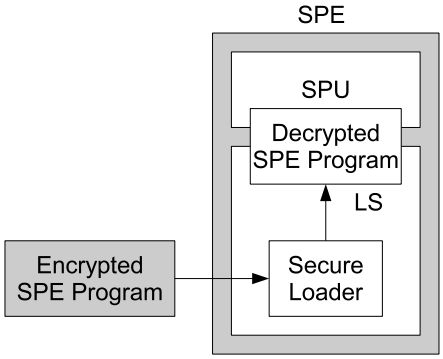

図 3 Secure Loader による安全なロード

カルストア（LS）と呼ばれるメモリをコア内部に持ち、LS はメインメモリから物理的に分離されている。SPE は LS にプログラムをロードし、DMA 転送を行ってメインメモリと LS の間で情報をやり取りしながら処理を行う。

3.2 Isolation モードを用いた安全な実行

SPE Isolation モードは OS 監視システムを安全に実行することを可能にする。Isolation モードは SPE が内部に持つ LS に対し PPE や他の SPE からのアクセスを禁止する。SPE 上で動かすプログラムや SPE 内部で処理されるデータは全て LS の内部に置かれるため、Isolation モードを用いれば LS 内部にある実行中のプログラムやデータを外部から書き換えることはできず、OS 監視システムの完全性を保証できる。また、実行中のプログラム内部で扱っている暗号鍵等の機密情報を読み出されることもないため、OS 監視システムの機密性も保証できる。

Isolation モードで動く OS 監視システムは Secure Loader [6] によって安全に SPE にロードされる。まず、暗号化された Secure Loader が SPE にロードされ、SPE はハードウェアに埋め込まれた暗号鍵を用いて Secure Loader の署名を検証した後、Secure Loader を復号化する。次に、図 3 のように Secure Loaser が暗号化された監視プログラムを LS にロードし、Secure Loader が持つ暗号鍵を用いて監視プログラムの署名を検証した後、監視プログラムを復号化する。

ハードウェアによって完全性が保証された Secure Loader が監視プログラムの完全性をチェックするため、攻撃者は改ざんした監視プログラムをロードすることができない。さらに、ディスク上に置かれている監視プログラムの機密性も保証される。監視プログラムを復号化する暗号鍵は Secure Loader が持ち、Secure Loader を復号化する暗号鍵はハードウェア内部にあるため、攻撃者は監視プログラムを復号化することができない。

Isolation モードで動いている SPE 上のプログラムの完全性と機密性は保たれるが、PPE は SPE 上の OS 監視システムを停止させることができる。これは PPE 上の OS がシステム全体を管理しているためである。しかし、OS 監視システムを停止されたとしても機密性を保つことができる。プログラムを停止して Isolation モードを解除すると、ハードウェアによって LS の内容がすべて消去されるためである。PPE から別のプログラムをロードして LS 上の機密情報を取得することはできない。

3.3 セキュリティプロキシ

外部のセキュリティプロキシから SPE 上の OS 監視システムに定期的にハートビートを送ることで、可用性を向上させる。PPE から SPE 内部のプログラムを終了させられるので、監視プログラム単体では可用性を保つことができない。ハートビートへの応答がなくなれば、OS 監視システムが停止されたと判断して、即座に対処を行うことができる。また、攻撃者に OS 監視システムを止めさせないようにするための抑止力にもなる。

正しい監視プログラム以外が応答を行えないようにするために、セキュリティプロキシと Isolation モードで動いている監視プログラムの間で暗号通信を行う。Isolation モードにより SPE 内部の暗号鍵を攻撃者は知ることができないので、攻撃者にネットワーク上を流れる暗号文を解読することは困難であり、正しい応答を返すことができるのは監視システムが動いている SPE のみであることが保証できる。

セキュリティプロキシは暗号に対して間違った応答を返されるか一定時間応答がなかった場合、監視対象マシンに出入りするパケットを遮断する。セキュリティプロキシは監視対象マシンに出入りするパケットを中継するため、すべてのパケットは必ずセキュリティプロキシを通る。パケットを遮断されると攻撃者はネットワークを経由して攻撃を継続することができなくなる。攻撃プログラムは動き続けるが、監視対象マシンはネットワークから遮断されているため、被害をこのマシンに留めることができる。一方、外部への攻撃を継続するためには監視システムを動かさざるを得ず、OS が改ざんされていないかチェックを継続することができる。

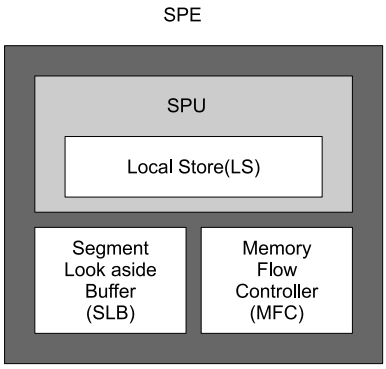

図 4　SPE の物理構成

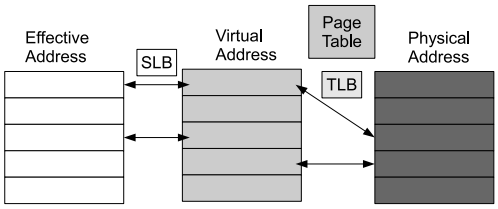

図 5　Cell/B.E. でのメモリ管理

4　実装

本システムを IBM が提供している Cell/B.E. SDK 3.1 と Cell/B.E. Security SDK を用いて実装した。現状ではハードウェアが提供する SPE Isolation モードを使用することができなかったため、Security SDK が提供している Isolation モードのエミュレーションを利用した。Secure Loader も Security SDK が提供しているものを利用した。また、ハードウェアの Isolation モードと同様に SPE を占有させるために、対象の SPE が OS によってスケジューリングされないようにした。

4.1　カーネル監視手法装

SPE にカーネルメモリへのアクセス権限を持たせるために、SPE の持つ Memory Flow Controller（MFC）の状態レジスタ 1 の Problem-State ビットをクリアした。MFC は図 4 のように各 SPE 内にあり、SPE がメインメモリに情報を送受信する際に DMA 転送を行う部分である。SPE から MFC のレジスタにデータを書き込むことで DMA 転送を行うことができる。

通常、SPE は PPE 上のプロセスのアドレス空間にしかアクセスできないので、SPE がカーネルアドレス空間にアクセスできるようにするために、SPE の持つ Segment Lookaside Buffer (SLB) にマッピングを登録する。Cell/B.E. ではカーネルメモリを含むメモリアドレスが実効アドレスで管理されている。実効アドレスは仮想アドレスにマップされ、仮想アドレスはメモリの物理アドレスにマップされている。図 4 のように各 SPE が持つ SLB は実効アドレスから仮想アドレスへの変換テーブルであり、SLB に実効アドレスからのマッピングを登録することでメインメモリにアクセスすることができる。

カーネルメモリからの DMA 転送と監視処理をオーバーラップさせるために、SPE 上の監視システムはダブルバッファリングを行う。DMA 転送を行うには時間がかかるが、転送の完了を待っている間に処理を

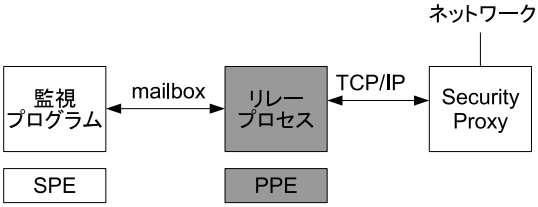

図 6 PPE を介したハートビート

行うことができる。具体的には、DMA 転送の完了を待っている間に直前に取得したメモリブロックに対して監視処理を行う。

監視処理の一例として、DMA 転送により System.map に定義されているカーネルメモリのテキストとリードオンリデータ領域のチェックサムを計算し、あらかじめ計算した値と比較する。

4.2 ハートビート

プロキシから SPE 上の OS 監視システムにハートビートを送るために、PPE 上のリレープロセスがパケットを中継する。SPE にプロトコルスタックを実装し、直接 NIC にアクセスすることで SPE が直接通信を行うことも可能 [4] だが、ネットワーク処理のために LS を大量に消費してしまう。

セキュリティプロキシと Cell/B.E. 搭載マシン間の通信には TCP/IP を用い、PPE と SPE の間の通信にはメールボックスを用いる。 メールボックスとは SPE の MFC が持つメールボックスチャンネルを介して 32bit のデータをやり取りする機能である。この機能を用いれば、SPE が Isolation モードでも PPE と SPE の間でデータを送受信することが可能である。SPE がメッセージを書き込むチャンネルと、SPE がメッセージを読み込むチャンネルが用意されている。

セキュリティプロキシはハートビートで送るメッセージを乱数で生成し、あらかじめ共有しておいた暗号鍵を用いて暗号化し、監視対象マシンに暗号文を送る。この暗号文は PPE で動くリレープロセスが一旦受け取り、SPE にメールボックスを用いて渡す。SPE はメールボックスの受信キューに暗号文がある場合、DMA 転送完了待ちの時に暗号文を受け取り、復号化と応答メッセージの生成を行う。作成した応答メッセージを暗号化し、メールボックスの送信キューが空になったら PPE に向けてメールボックスを用いて送信する。PPE は受け取った暗号文をセキュリティプロキシに送り、セキュリティプロキシは暗号文を復号化し、応答メッセージをチェックする。

4.3 スケジューリング

SPE Observer は監視システムを２通りの方法で動かすことができる。１つは SPE を１つ占有し、常時監視を行う方法である。この方法を用いると、起動のオーバーヘッドがかかるのが最初だけだが、SPE １つ分システム全体の性能が低下する。

もう１つは定期的に監視システムを動かして、監視を行っていない時は SPE を解放する方法である。空き SPE があればその SPE を用いて監視を行い、空き SPE がなければいずれかの SPE 上で実行中のプログラムをサスペンドして、その SPE を一時的に使用する。実行が終わればサスペンドしたプログラムをレジュームする。この方法を用いると、毎回起動のオーバーヘッドがかかるが、システム全体の性能低下を抑えることができる。

5 実験

実験には、Sony Computer Entertainment 社製の PLAYSTATION3 80GB モデルを用いた。このマシンで使用可能な SPE 数は６つである。また、Fedora 9 をインストールし、Linux カーネルバージョン 2.6.27 を使用した。

5.1 監視の実行時間

カーネルを監視するために、カーネルメモリを読み込みの時間を測定した。この実験では、カーネル本体と同じサイズである 12MB を読み込んだ。表１に、SPE からカーネルメモリを読み込んだ場合と PPE から読み込んだ場合にかかる時間を示す。 メモリ取得は SPE で行った方が速いことが分かる。１回のメモリ取得を行う際に、SPE では DMA を用いてまとめて取得しているが、PPE では１ワードずつ取得するためである。

表 1 監視にかかる時間

動作コア	時間（msec）
SPE	7
PPE	72

表 2 バスへの影響評価

SPE 数	時間（秒）
1	1.91
6	1.91

5.2 カーネル改ざんの検出

SPE Observer を用い、カーネルのテキスト領域とリードオンリデータ領域のチェックを行う OS 監視システムを動作させ、カーネルの改ざんを検出できるかどうか調べた。チェックサムを計算したのと同じカーネルと、カーネルの名前を書き換えたもの、システムコールを追加したカーネルについてチェックを行った。その結果、同じカーネル以外はチェックサムの結果が異なり、改ざんの検出ができた。

5.3 バス帯域への影響

OS 監視システムは DMA 転送を行っているので、他の SPE プログラムも DMA 転送を行っていた場合、バスのトラフィックが混雑してシステム全体の性能が低下する可能性がある。そこで、６つの SPE で同時に 120MB のメモリ転送を行うプログラムを動かし、メモリ転送にかかる時間を測定した。比較のために、１つの SPE だけで動かした場合も測定した。１回の DMA 転送では、最大の 16KB を転送するように指定した。 表 2 に、SPE6 つが DMA 転送を行った場合と、SPE1 つが DMA 転送を行った場合の実行時間を示す。6 つの SPE が同時に DMA 転送を行っても実行時間は変わらなかった。これより監視プログラムを動かしてもバス帯域には影響が出ないと考えられる。

5.4 SPE を占有する影響

OS 監視システムが SPE を１つ占有した場合、6 つの SPE を使うアプリケーションの実行に影響を及ぼす。SPE を 6 つ使うことを想定したアプリケーションと OS 監視システムを同時に動かした場合、アプリケーションの実行にどのような影響があるか調べた。アプリケーションとして 5.3 節で使用したプログラムを 35 回ループさせた。 表 3 に、OS 監視システムを動かした場合のアプリケーションの処理時間と、動かしていない場合の処理時間を示す。OS 監視システムを動かすと、アプリケーションは５つの SPE で動かさなければならないため、性能が低下している。

表 3 SPE 占有時のアプリケーションへの影響

	実行時間（秒）
システム起動時	80.7
システム非起動時	66.8

利用可能な物理 SPE より多くのコンテキストを作成した場合、Linux カーネルがコンテキストを切り替えながら実行を行う。このコンテキスト切り替えは占有状態にない SPE 間で平等に行われる。今回は SPE の０番を OS 監視システム用に占有したため、アプリケーションの切り替えは１番から５番の SPE で順番に行われていた。OS 監視システムなしでは６つの SPE を使い、のべ 66.8×6 秒の処理を行っている。これを５つの SPE で行うと 80.2 秒になる計算である。OS 監視システムありとの差の 0.5 秒がコンテキスト切り替えのオーバーヘッドと考えられる。これにより、性能低下はほぼ１つのコア分である。

6 関連研究

SPE Isolation モードを用いた安全なデータ解析 [9] が提案されている。この研究では、ボランティアのマシンでデータ解析を行うのに SPE Isolation モードを用いる。ボランティアに渡されるデータは暗号化されており、解析プログラムがデータを復号化する。SPE でデータ解析を行った後に解析結果を暗号化して送っている。データが暗号化されているので、ボランティアのマシンのユーザは扱っているデータの内容を読むことができず、データのプライバシを守ることができる。

SPE Isolation モードを用い、PPE 側で実行するアプリケーションの完全性を SPE から保証する Code Verification Service [6] が提案されている。PPE がアプリケーションをシステムメモリにロードしたあ

と、Isolation モードで動いている Code Verification Service にアプリケーションの検証を依頼する。アプリケーションがロードされているシステムメモリを DMA 転送し、SPE 内の暗号鍵を使ってアプリケーションの署名を検証する。そして、Code Verification Service から改ざんされていないという結果が返されたら、PPE はアプリケーションを安全に実行することができる。

Intel や AMD の CPU は 80386 以降、System Management Mode（SMM）と呼ばれるモードを提供している。SMI と呼ばれる最高優先度の割り込みが発生した時に、SMRAM 上のプログラムが実行される。SMRAM は通常のモードではアクセスすることができないメモリ領域である。このモードを用いれば OS や他のアプリケーションから実行を邪魔されることなく、安全にソフトウェアを実行できる。ただし、基本的に BIOS からのみ利用可能である。

Flicker [5] は Intel TXT や AMD SVM などのハードウェア機構を使ってセキュリティの必要なアプリケーションを安全に動作させることを可能にする。このようなアプリケーションを動かす際には、すべてのコアでのプログラム実行を OS も含めて停止させ、TPM を用いてロードされるアプリケーションの完全性を検証する。そして、割り込みや DMA などを禁止した状態でアプリケーションを実行する。Flicker ではセキュアアプリケーションの実行中は他のプログラム実行が完全に停止するため、常時動作させる必要がある監視プログラムには向かない。また、SMM から攻撃が可能であることが指摘されている [10]。

7 まとめと今後の課題

Cell/B.E. の SPE 上で安全に OS 監視を行うことができる SPE Observer を提案した。このシステムを使用することにより、OS カーネルが書き換えられていないかチェックすることができる。SPE Isolation モードを用いることにより、OS 監視システムの完全性と機密性が保証される。また、外部のセキュリティプロキシからハートビートを送ることにより監視システムの可用性を向上させることができる。SPE Observer を PLAYSTATION3 に実装し、カーネルの整合性をチェックできるようにした。実験により、OS の改ざんを検知でき、SPE を 1 つ占有した場合でも性能への影響は限定的であることがわかった。

今後の課題として、現在は SPE Isolation モードのエミュレーションを用いているため、ハードウェアの機能を用いて実現することが挙げられる。また、必要な時だけ OS 監視システムを動かせるようにする実装を完成させ、性能への影響を調べる必要がある。PPE 上の攻撃者に SLB を書き換えられてしまうと SPE はカーネルが使っているのとは異なるメモリをチェックしてしまうので、SLB の改ざんも検出できるようにする必要がある。

謝辞

本研究の一部は、科学技術振興機構 戦略的創造研究推進事業（CREST）研究領域「実用化を目指した組み込みシステム用ディペンダブル・オペレーティングシステム」、および、科学研究費補助金（課題番号：21500039）による。

参 考 文 献

[1] CVE: CVE-2008-2004, 2008.
[2] CVE: CVE-2008-4405, 2008.
[3] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, In Proc. Symp. Network and Distributed System Security, 2003, pp. 191–206.
[4] Kawamura, Y., Yamazaki, T., Ishiwata, T., Horie, K., and Kyusojin, H.: Network Processing on an SPE Core in Cell Broadband Engine, In Proc. Symp. High Performance Interconnects, 2008, pp. 119–128.
[5] McCune, J. M., Parno, B., Perrig, A., Reiter, M. K., and Isozaki, H.: Flicker: An Execution Infrastructure for TCB Minimization, In Proc. European Conf. Computer Systems, 2008, pp. 315–328.
[6] Murase, M., Plouffe, W., Shimizu, K., and Sakamoto, M.: Effective Implementation of Cell Broadband Engine Isolation Loader, In Proc. Computer and Communications Security Conf., 2009.
[7] SCEI: Cell Broadband Engine Architecture Version 1.02, 2007.
[8] Trusted Computing Group: Trusted Platform Module, http://www.trustedcomputinggroup.org/.
[9] Wang, H., Takizawa, H., and Kobayashi, H.: A Performance Study of Secure Data Mining on the Cell Processor, In Proc. Symp. Cluster Computing and the Grid, 2008, pp. 633–638.

[10] Wojtczuk, R. and Rutkowska, J.: Attacking Intel Trusted Execution Technology, Black Hat DC, 2009.