九州大学学術情報リポジトリ
Kyushu University Institutional Repository

情報社会を支えるディペンダブル・プロセッサ

井上, 弘士
九州大学大学院システム情報科学研究院

https://hdl.handle.net/2324/9185

出版情報：情報処理学会研究報告, 2007-SLDM-131. 2007 (105), pp.1-6, 2007-10. 情報処理学会
バージョン：
権利関係：

情報社会を支えるディペンダブル・プロセッサ

井上弘士†

本稿では，アーキテクチャ・レベルでの安全性向上を目的としたセキュア・プロセッサに関する研究事例を紹介する．近年，コンピュータ・ウィルスや情報漏洩が極めて深刻な社会問題となっている．これまでに，ネットワーク・レベルやシステム・ソフトウェア・レベルにて多くの安全性向上技術が開発されており実用化も進んでいる．しかしながら，依然としてコンピュータ・システムに対する脅威は増加の一途を辿っている．1970 年代初頭にマイクロプロセッサが開発されて以来，トランジスタ集積度の向上に伴い順調な性能向上を達成してきた．また，携帯機器の普及に伴い，低消費電力化や低消費エネルギー化も進んでいる．しかしながら，安全性向上に関する議論は少なく，2000 年以降になって本格的にアーキテクチャ・レベルで安全性を考慮する必要性が認識されるようになった．今後，高性能化や低消費電力化と同様に，安全性向上技術は極めて重要な設計制約となる．

Dependable Processors for Social Information Infrastructures

KOJI INOUE†

This paper introduces architectural supports to improve the efficiency of computer security. In the social information infrastructures, we exactly face to “Security Problem” such as computer viruses and information leaks. Although a number of techniques to improve security efficiency, which focus on network and system software components, have so far been proposed, still many threats exist. Since 1970s, microprocessors have made incredible progress in terms of performance. In addition, from 1990s, many techniques to reduce power or energy consumption have been developed. However, a few discussions for computer security at the processor level have done. Now, it is the time to start considering, how we can improve the security efficiency by means of providing architectural supports.

1. はじめに

1970 年代初頭に開発されて以来，プロセッサは現在までに目覚しい発展を遂げてきた．微細加工技術の順調な進歩を背景に，並列処理や投機実行，オンチップ・キャッシュの大容量化など，豊富なトランジスタ資源を有効利用することで劇的な性能向上を達成した．また，1990 年代中頃から PDA や携帯電話といったモバイル機器が広く普及するようになり，バッテリ寿命の延長に対する要求が高まった．その結果，プロセッサの低消費電力化/低消費エネルギー化に関する研究・開発が加速し，現在では電源電圧の動的最適化など様々な技術が実用化されている．

このような技術革新により，プロセッサの応用も広がった． PC や携帯電話，家電製品はもちろんのこと，自動車や航空機に搭載された制御システム，電子マネーを対象としたアプリケーションなど，人

† 九州大学大学院システム情報科学研究院　〒819-0395 福岡県福岡市西区元岡 744 番地

命や財産に直接関係する場合もある．これほどまでに現代社会に深く浸透したプロセッサであるが，はたして我々は安心して利用できるのであろうか？多くのユーザは，「正しく操作すれば，コンピュータは自分が指示した仕事だけを正確に処理してくれる」と信じている．しかしながら，近年，この大前提が成り立たない状況が発生している[6].

第 1 の理由は，プロセッサの信頼性の低下である．例えシステムに不具合がなくとも，様々な外的要因により「正しくプログラムを実行する」ことができない場合がある．その主な原因として，LSI 内部における一時的な故障の発生が挙げられる．予期しない不都合な物理的現象によりプロセッサが誤動作し，引いては，誤った実行結果を出力することになりかねない．以前より LSI の故障は問題視されていた．しかしながら，近年の超微細化やクロックの高速化，低電源電圧化などに伴い十分な設計マージンを確保することが難しくなり，故障発生確率が高くなってきている．

第 2 の理由は，第三者による悪意ある攻撃の存在である．現在我々は，整備されたネットワーク環境を活用して遠隔地の計算資源やデータベースへ容易にアクセスできる．しかしながら，これは，自分のコンピュータ・システムが攻撃対象になり得ることを意味する．例えば，コンピュータ・ウィルスはネットワークを介して世界中に広がり，極めて深刻な社会問題を引き起こしている．例えば，コンピュータ・ウィルスやワームなどの悪質プログラムは深刻な社会問題を引き起こしており，世界で発生したウイルス被害額は 2003 年で 135 億ドル（およそ 1.5 兆円）にもなる[4]．また，利用者が気づかないうちに攻撃を受け，コンピュータが不当に利用される場合や，秘密データが改竄/盗聴されるなどの被害もある．

このような背景を基に，近年，「信頼性」や「安全性」を考慮したプロセッサの研究・開発が活発化している．高性能化や低消費電力化のためだけでなく，「安心して利用できる道具」としての極めて本質的な要件を満足すべく，プロセッサ・アーキテクチャを考え直そうという流れである．本稿では後者の着目し，プロセッサ・アーキテクチャ・レベルでの安全性向上技術に関して述べる．

以下，第 2 節ではアーキテクチャ・レベルでの安全性向上技術の重要性を述べる．次に，第 3 節では，不正プログラム実行に焦点を当て，その解決策を体系的に整理する．そして，第 4 節ならびに第 5 節では，これまでに提案したセキュア・プロセッサ方式を紹介する．最後に第 6 章で簡単にまとめる．

2. なぜ，アーキテクチャ・レベルでセキュリティ？

これまでに，データの暗号化やインターネットを介した不正アクセスの検出，ソフトウェアによるウイルス検索など，様々な安全性向上技術に関する研究・開発が進められてきた．そして，これらの多くがアルゴリズム・レベル，ネットワーク・レベル，OS などのソフトウェア・レベルでの議論であった．しかしながら，依然として安全性に関する脅威は後を絶たないのが実状である．

ここで，携帯電話やパソコンなどのコンピュータ・システムに対する遠隔攻撃（破壊などの物理的な攻撃は対象外とする）に焦点を当てる．この場合，システム階層の上位レベルにおける防御策のみで十分であろうか？例えば，コンピュータ・ウイルスと言えども，それを最終的に実行するのはハードウェアである．また，攻撃対象となる重要な秘密情報を記憶し処理を施すのもプロセッサやメモリといったハードウェアである．一方，システムの安全性を向上するには，様々な攻撃からの防衛策を考案するのはもちろんのこと，性能や消費電力も含めたコストとのトレードオフを考慮することが重要となる．高い安全性を確保できる方式を考案したとしても，その実現において極めて高いコストを要する（例えば，プログラム実行時間や消費電力が数倍になるなど）場合には適用することが難しい．そのため，安全対策を施すこと事態が難しくなり，より危険性を高める結果となりかねない．

すなわち，安全性を向上するためには，「安全性とそれを得るためのコスト（性能や消費電力など）のトレードオフを考慮し，適切なアプローチを採る」必要がある．そして，これを実現するためには，

- 実際に処理を行うハードウェア・レベルにて，
- プログラム実行の振舞いを考慮した安全性向上策を実現

しなければならない．これら 2 つの要件を満足するためには，従来のネットワークやソフトウェア・レベルだけでなく，アーキテクチャ・レベルで安全性を考えることが極めて重要となる．

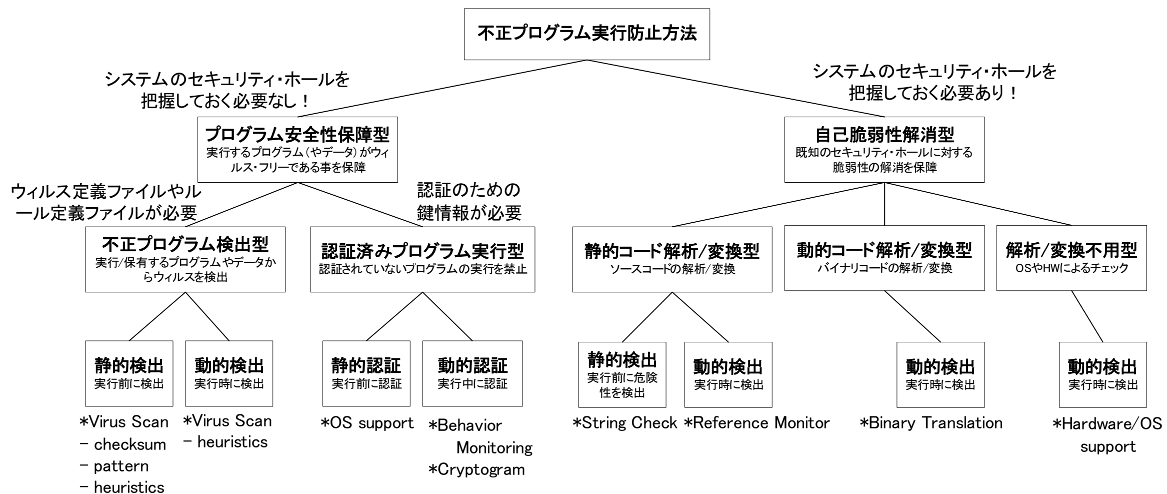

図１：不正プログラム実行防止技術の分類

3. 不正プログラム実行防止技術の分類

これまでに様々な不正プログラム実行防止技術が考案されてきた．また，その一部はすでに実用化されている．不正プログラム実行防止技術は主に，図 1 に示すように 2 つの方式に大別できる[5].

- 自己脆弱性解消型：システム内部に存在するセキュリティ・ホールを活用する不正プログラムの実行を防止する．つまり，静的もしくは動的にセキュリティ・ホールへの攻撃（または攻撃可能性）を検出し，それに対する防御を行う．例えば，バッファ・オーバフロー脆弱性に基づくスタック・スマッシングを検出する方法などが多く提案されている．本方式の利点は，対処済みのセキュリティ・ホールを攻撃する全ての不正プログラムに関して，その実行を防止できる点である．当然，防御の対象となるセキュリティ・ホールは既知でなければならないが，システム内部の全てのセキュリティ・ホールを事前に把握する事は極めて難しい．
- プログラム安全性保障型：実行対象となるプログラムが不正コードを含まない事を保障する（安全性を保障する）．つまり，システム内部に如何なるセキュリティ・ホールが存在するとしても，実行対象となるプログラムが安全であれば問題は発生しないという考えに基づいている．本方式では，システム内部に存在するセキュリティ・ホールを事前に把握しておく必要がないといった利点がある．実際，実用化されている不正プログラム実行防止技術の多くは本方式に分類される．

プログラム安全性保障型は，更に以下のように分類できる．

- 不正プログラム検出型：ウィルス定義ファイルやルールファイル等に基づき，システム内部に不正プログラムが存在しないか検査する．本方式ではアプリケーション・プログラムに対して特別な処理を加える必要がないため，その適用は比較的容易である．実際，ウィルス・スキャン等の技術は実用化が進んでおり，すでに多くのコンピュータ・システムに搭載されている．しかしながら，本方式の最大の欠点は未知の不正プログラム（ウィルス定義ファイルやルールファイルに登録されていない不正プログラム）の検出ができない点にある．
- 認証済みプログラム実行型：アプリケーション・プログラムの安全性が保障されたコードのみを実行する．つまり，プログラムを実行するコンピュータ・システムにおいて，安全であると認証されたプログラムのみが実行可能となる．したがって，未知の不正プログラムに対しても対応することができる．ただし，プログラムの実行に関して，静的または動的なプログラム認証処理が必要となる．

4. 不正プログラムの実行を防止する！～バッファオーバフロー攻撃の動的検出～

近年，バッファ・オーバフローの脆弱性を活用した悪質プログラムによる被害が急増している．例えば，代表的なものとして 2001 年に猛威を振るった Code Red や，2003 年の Blaster などがある． 悪質プログラムは，攻撃対象となるコンピュータが正規アプリケーションを実行している最中にバッファ・オーバフローを引き起こさせ，強制的にプログラムの実行制御を乗っ取る．したがって，特権モードでの実行中にバッファ・オーバフローが発生した場合，悪質プログラムは特権モードで実行されることになる．その結果，ファイルの削除や改ざんが可能となり多大なる被害をもたらす．例えば，右図に示すように，CERT によって発せられた勧告(1996～2001 年)の内，バッファ・オーバフローに関連するものが高い割合を占めている．

多くの悪質プログラムは，関数呼び出し後にバッファ・オーバフローを引き起こしてスタックを破壊する(スタック・スマッシングと呼ばれる)．そして，関数呼出し側への戻りアドレスを悪質プログラム・コードの先頭アドレスへと改ざんすることで，プログラムの実行制御を乗っ取る．図 2 を用いてバッファ・オーバフローによる攻撃の様子を簡単に説明する．ここでは，C 標準ライブラリである strcpy を用いて文字列コピーを行う関数 g が，関数 f によって呼出される場合を想定している．通常，関数 g が呼出された際，関数 f への戻りアドレスをスタックに保存する．そして，関数 g での処理終了後，この戻りアドレスを PC に復元する事で関数呼出し側へと実行制御が移る．これに対し，バッファ境界をチェックしない文字列コピーによりスタックの破壊（スタック・スマッシング）が発生した場合，スタック領域に対して悪質プログラム・コードが上書きされる．また，関数 f への戻りアドレスが悪質コードの先頭アドレスへと改ざんされる．そして，呼出し元関数 f へ復帰する際には改ざんされた戻りアドレスが用いられ，その結果としてスタック内部の悪質プログラム・コードへと実行制御が移る．

このようなバッファ・オーバフローによる実行制御の乗っ取りを動的に検出するアーキテクチャ・アプローチとして，**セキュア・キャッシュ(*SCache*)** を提案している[1][2]．通常，関数呼出し時にスタック領域へプッシュされる戻りアドレスは，一旦キャッシュにストアされる．また，呼出し元関数へ復帰する際，プロセッサはキャッシュから戻りアドレスをポップする．スタック・スマッシングによるプログラム制御の乗っ取りにおいて，その本質的な問題点は戻りアドレスが改ざんされることにある．そこで SCache では，図 3 で示すように，戻りアドレスがストアされる際，読出し専用の複製ライン(**レプリカ・ライン**と呼ぶ)を同一セット内に作成する．その後，戻りアドレスをロードする時，スタック領域から読出される値と，レプリカ・ラインの値を比較する．もし，比較結果が一致であれば戻りアドレスの安全性が保障され，不一致の場合にはスタック・スマッシングの発生を検出する．

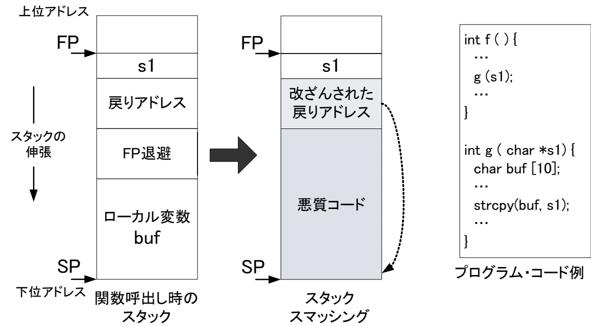

図 2：スタック・スマッシング

SCache のシミュレータ作成，ならびに，キャッシュ構成要素の実設計を行い定量的評価を実施した．その結果を図 4 に示す．ここで，

$$Vulnerability = (Nv\text{-}rald / Nrald) * 100 \quad (1)$$

とする．*Nrald* はプログラム実行における IRA ロードの総数である． ここで IRA(Issued Return Address)ロードとは，キャッシュ・メモリに対して発行された戻りアドレス・ロードの事である．また，*Nv-rald* は戻りアドレス改ざんを検出できない(安全性を保障できない)IRA ロード総数を表す．多くのプログラムにおいて，99%以上の戻りアドレス書込みを保護できることが分かった．また，その時の性能ならびに消費エネルギー・オーバヘッドは，それぞれ，0.5%ならびに 20%以下であった．なお，図において各評価対象モデル名の最後の数字は生成される複製データの数，LRU ならびに MRU は複製データの配置アルゴリズムを表している．

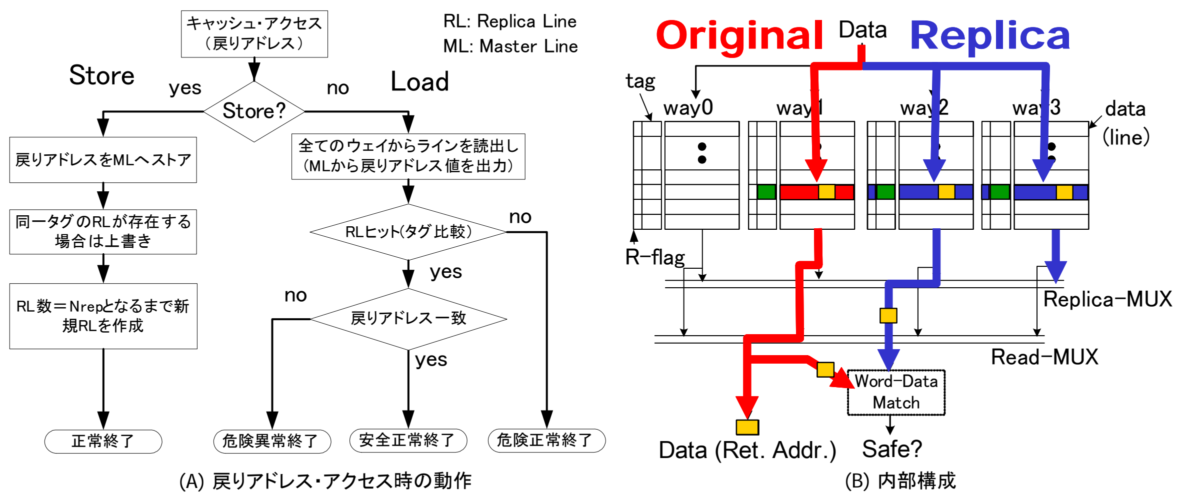

図 3：セキュア・キャッシュの動作と内部構造

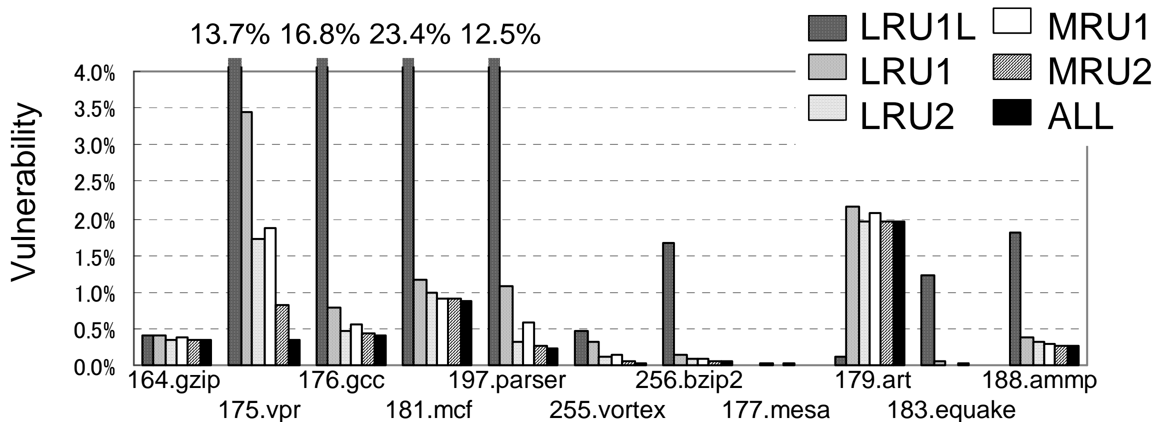

図 4：セキュア・キャッシュの安全性

5. 不正プログラムの実行を防止する！～動的プログラム認証技術～

攻撃対象となるシステムの脆弱性が既知の場合には，前節で説明した SCache のようにハードウェアで対策を施すことができる．しかしながら，システム開発者にとって未知の脆弱性に関しては対応することができない．この問題の解決策として，実行対象となるプログラムが不正コードを含まない事を保障（安全性を保障）し，このような認められたプログラムのみを実行可能とする方法が挙げられる（いわゆる，ホワイトリスト方式）．つまり，システム内部に如何なるセキュリティ・ホールが存在するとしても，実行対象となるプログラムが安全であれば問題は発生しないという考えに基づく．この方式においては，「如何にして対象プログラムがウィルスフリーである事を検査するか？」，また，「如何にしてプロセッサが承認済みプログラムコードであることを認知するか？」が重要となる．前者に関しては，アプリケーション・プログラム発行者が保障する，または，既存のウィルス検索ソフト等によりチェックする方法が考えられる．しかしながら，後者に関しては，現在のプロセッサで実現することは難しい．プロセッサから見たプログラム実行とは単なる「命令シーケンスの処理」にしかすぎないためである．つまり，悪質プログラムの実行においても，安全であると保障された正規プログラムの実行と同様に「ただの命令シーケンスの実行」としてしか認識していない．

そこで，性能オーバヘッドが小さく，かつ，ソー

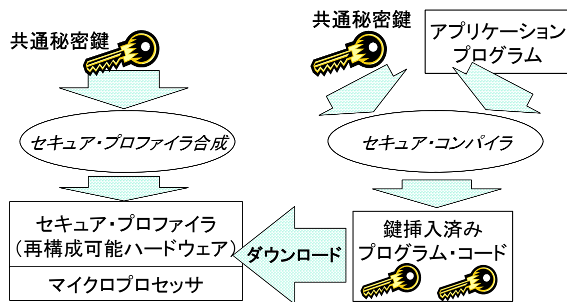

図 5：実行振舞いを鍵情報とするプログラム認証

スコードに依存しないプログラム認証方式として，実行の振舞いを鍵情報とする動的プログラム認証方式を提案している[3]．提案方式の基本動作を図 5 に示す．ここでは，利用者側とアプリケーション発行側で共通の秘密鍵を保有していると仮定している．アプリケーション発行側では，この秘密鍵から決定される「プログラム実行の振舞い」を実現するオブジェクト・コードを生成する．一方，利用者側では，秘密鍵から決定される「プログラム実行の振舞い」を検出するための専用プロファイラを生成し，再構成可能ハードウェア上に実装する．そして，アプリケーション・プログラム実行時，このプロファイラによって常に実行の振舞いを監視する．鍵としての実行の振舞いが検出できなかった場合には即座にプロセッサに実行停止割り込みを発行する．

ここでは，鍵情報となる実行の振舞いの一例としてメモリアクセス・パタンに着目する．具体的には，ある特定アドレスへのロード命令を「鍵ロード命令」と呼び，「N 命令毎に鍵ロード命令が必ず実行される」という実行の振舞いを鍵情報とする．

1. アプリケーション発行者は，プログラムのコンパイル時，必ず N 命令毎に鍵ロード命令が実行されるようコードを生成する．
2. 利用者側では，プログラムの実行を開始する前に鍵検出ハードウェア（再構成可能ハードウェア）を準備する．ここでは，実行命令カウンタ，ならびに，メモリ参照モニタが必要となる．
3. プログラム実行を開始すると同時に，鍵検出ハードウェアによって実行の振舞いをモニタする．もし，N 命令毎に鍵ロード命令が実行されない（つまり，特定アドレスへのロードが発生しない）場合は不正プログラムの実行と見なす．

本方式では，コンパイル時にプログラム実行振舞いを制御する必要がある．これを可能にするため，基本ブロックのサイズを統一する．これに伴い，性能に関するオーバヘッドが発生する．性能評価結果を行った結果，統一基本ブロック・サイズを 5 命令に統一した場合の性能オーバヘッドは 9%程度であることが分かった．不正プログラムの実行防止を優先する場合には 10%以下の性能低下は許容範囲であると考える．

6. おわりに

本稿では，次世代の高度情報化社会を支える要素技術として，プロセッサ・アーキテクチャ・レベルでの安全性向上策を紹介した．我々の日常生活は，よりコンピュータ・システムに依存したものとなるであろう．このような将来を見据え，アーキテクチャ・レベルのみならず，システム階層全体における統合的な安全性向上技術の確立が重要になると考える．

謝辞

本講演の機会を与えて頂いた九州大学システム LSI 研究センターの石原亨准教授，ならびに，SLDM 研究会の皆様に心より感謝いたします．なお，本研究は一部，科学技術振興機構さきがけプロジェクト「情報基盤と利用環境」領域，ならびに，文部省科学研究費補助金(課題番号： 17680005）による．

参 考 文 献

[1] K. Inoue, ``Return Address Protection on Cache Memories,'' *IEICE Transactions on Electronics,* Vol. E89-C, No.12, pp.1937-1947, Dec. 2006.

[2] Koji Inoue, "Lock and Unlock: A Data Management Algorithm for A Security-Aware Cache," *IEEE International Conference on Electronics, Circuits and Systems (ICECS'06)*, pp.1093-1096, Dec. 2006.

[3] Koji Inoue, "Supporting A Dynamic Program Signature: An Intrusion Detection Framework for Microprcessors," *IEEE International Conference on Electronics, Circuits and Systems (ICECS'06)*, pp.160-163, Dec. 2006.

[4] IPA(Information-Technology Promotion Agency, Japan)の「国内・国外におけるコンピュータウイルス被害状況調査(2004 年 4 月)

[5] 井上弘士，岩佐崇史，``実行の振舞いを鍵情報とする不正プログラムの動的検出方式,''情報処理学会研究報告 2005-ARC-164, pp.25-30, 2005 年 8 月

[6] 井上弘士，“信頼性・安全性とプロセッサ（新世代マイクロプロセッサ・アーキテクチャ後編），” 情報処理，Vol.46, No.11，通巻 489 号，2005 年 11 月．