# メモリ参照パタンを鍵情報とする セキュア・プロセッサ・システムに関する一考察

# A secure processor system with key information of memory reference pattern

岩佐 崇史[†] 井上 弘士[‡ ††]

福岡大学大学院工学研究科電子工学専攻[†] 福岡大学工学部電子情報工学科[‡]
科学技術振興機構さきがけ[††]

Takafumi IWASA[†] Koji INOUE[‡ ††]

Dept. of Electronics Engineering, Fukuoka University[†]

Dept. of Electronics Engineering and Computer Science, Fukuoka University[‡]
Precursory Research for Embryonic Science and Technology[††]

## 1 はじめに

近年、コンピュータ・ウイルスなどの悪質プログラム(以下、不正プログラムと呼ぶ) による被害が深刻な問題となっている。このような不正プログラムは、ユーザが気づかないうちにコンピュータ内部に進入し、突然暴走することでその被害を拡大する。今後、社会システムの更なる情報化を進めていくためには、不正プログラムの検出技術ならびに実行防止技術の確立が必要不可欠となる。そこで我々は、不正プログラム問題の解決策として「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。本方式では、プログラム発行者と利用者が共通秘密鍵を有する場合を前提とし、プログラムのコンパイル時に鍵情報を実行振舞いとしてオブジェクト・コード中に挿入する。そして、当該プログラムが実行される際、「鍵情報としての実行の振舞い」を常に監視し、鍵となる振舞いが検出されなかった場合にはこれを不正プログラムと判断する。つまり、プログラム実行という最終段階に鍵情報を適用することで、静的プログラム認証では解決できない不正プログラムの突然の暴走も防ぐことが可能となる。

このような動的プログラム認証方式では、如何にして「コンパイル時におけるプログラム実行振舞いの制御可能性」を高めるかが極めて重要となる。通常、プログラム実行の流れは入力データに依存するため、コンパイル時に実行の振舞いを完全制御することは難しい。そこで、本研究では、基本ブロック・サイズの統一による実行振舞い制御可能性を検討する。また、基本ブロック・サイズを統一した場合に発生するコード・サイズの増大やそれに伴う性能低下に関して定量的評価を行う。

## 2 メモリ参照パタンを鍵情報とする動的プログラム認証

### 2.1 システム概要

現在、不正プログラムを検出する方法としては、ウィルス定義リストに基づくパターンマッチング方式が代表的である。しかしながら、この方法では、リストに登録されていない未知の不正プログラムには対応できないといった問題がある。これを解決する手段としては、プログラム認証方式が考えられる。正規アプリケーションである事が保障されている場合にのみその実行を許可することで、不正プログラムの暴走を阻止できる。従来のプログラム認証は、OS 等のシステム・ソフトウェアによる証明書の確認が一般的であった。しかしながら、最近ではシステム・ソフトウェアを改ざんする悪質な不正プログラムも出現している。また、プログラム認証は実行開始前(静的)に実施されるため、正規アプリケーションの実行中に不正プログラムが突然暴走し始めた場合には対処できない。

そこで、現在我々は「実行の振舞いを鍵情報とする動的プログラム認証方式」を提案しており、本稿では実行の振舞いとしてメモリ参照パタンに着目する。具体的には、鍵情報となるメモリ参照パタンを生成するための専用ロード命令(以下、鍵ロード命令と呼ぶ)を用意する。そして、アプリケーション発行側では、実行時に定められた時間間隔で鍵ロード命令が発行されるようオブジェクト・コードを生成する。一方、利用者側では、プログラム実行中に鍵ロード命令による参照パタンを採取し、鍵としての振舞いが検出されない場合には実行そのものを停止する。よって、正規プログラムの実行を不正プログラムに乗っ取られた場合、定められた時間間隔での鍵ロード命令参照が発生しないため、実行を停止して被害を食い止めることができる。

### 2.2 基本ブロック・サイズ統一

第 2.1 節で説明した動的プログラム認証方式では、ある時間間隔で鍵ロード命令が実行されるよう正確に制御できなければならない。しかしながら、プログラム実行の流れは入力データに依存し(分岐結果など)、さらには、各基本ブロックのサイズが異なるため、静的な命令スケジューリングによる鍵ロード命令の挿入は難しい。

この問題の解決策として、基本ブロック・サイズの統一が考えられる。つまり、全ての基本ブロックに関して実行命令数を同一にすることで、鍵ロード

命令出現タイミングの制御可能性を高める。具体的には、アセンブリ・コード・レベルで各基本ブロックの大きさを抽出する。そして、最もサイズの大きな基本ブロック(Max1 と呼ぶ)を検出し、他全ての基本ブロックも Max1 と同じサイズに変更する(nop 命令を挿入)。このように、全ての基本ブロックの大きさを統一することで、鍵ロード命令出現タイミングの可制御性は極めて高くなる。しかしその反面、無駄な nop 命令の挿入に伴いコード・サイズが大きくなるという問題が生じる。その解決策としては、N(>1)番目に大きな基本ブロック(MaxN と呼ぶ)を基準とし、それより小さな全基本ブロックの大きさを MaxN と同一にする方法が考えられる。例えば N が 2 の場合、Max1 を除く全基本ブロックは Max2 のサイズと等しくなり、コード・サイズ増加率を抑えることができる。しかしながら、Max1 とそれ以外の基本ブロックの間では実行タイミング制御が難しくなるため、Max1 に対して鍵ロード命令を挿入できない。つまり、N の増加に伴い鍵ロード命令を挿入可能な基本ブロック数が減少する。鍵ロード命令が挿入されていない基本ブロックの実行においては、動的プログラム認証を実施できない。したがって、N の値を大きくすることでコード・サイズ増加率は削減できる反面、安全性が低下することになる。

## 3 評価

基本ブロックサイズの統一に伴うコード・サイズ増加率、実行時間増加率、ならびに、安全性の度合い(全実行時間において、鍵ロード命令挿入可能な基本ブロック実行が占める割合)を評価するため、ベンチマーク・プログラムを用いたシミュレーションを行った。具体的には、ベンチマーク・プログラムにおいてサイズが最も大きい上位 10 個の基本ブロックを基準(つまり、1≦N≦10)とし、それぞれに関する実行コードを生成した。なお、本評価では SimpleScalar/ARM[1]によるサイクル・レベル・シミュレーション(イン・オーダ実行)を行っており、ベンチマーク・プログラムには SPECint95[2]の 129.compress を用いた(train 入力を使用)。

基本ブロックサイズの統一基準を Max1 から Max10 まで変更した再のコードサイズならびに実行時間(所要サイクル数)を図 1 と図 2 に示す。全ての結果は、サイズ統一を実施しない場合の結果で正規化している。これらの実験結果より、最も大きな基本ブロック(Max1)を基準とした場合、コードサイズは約 6 倍となり、実行時間は約 3.5 倍長くなっていることが分かる。しかしながら、2 番目に大きな基本ブロック(Max2)を基準にすることで、これらの増加率は大幅に改善されている(コードサイズは約 3.8 倍、実行時間は約 2.3 倍)。一方、全実行時間において、鍵ロード命令挿入可能基本ブロック(サイズを統一した基本ブロック)の実行が占める割合を図 3 に示す。この図より、Max1 の場合と同様に、Max2 を選択した場合でもほぼ 100%の実行時間を同一サイズ基本ブロックの実行で占めることができている。これは、使用したベンチマーク・プログラムにおいて、Max1 の実行頻度が極めて低かったためである。また、N を増加させるに従ってコードサイズや実行時間オーバヘッドを削減できている反面、安全性が低下しており、これらはトレードオフ関係にあることが分かる。以上より、ある程度の安全性を確保しつつ、コードサイズや実行時間の増加を抑える場合には Max7 程度が適切であると考える。

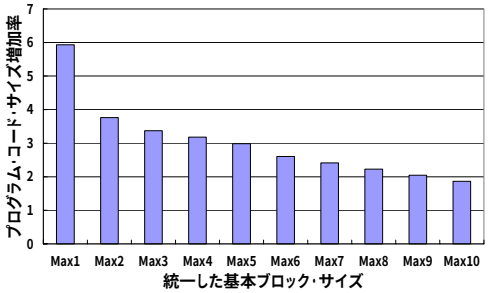


図 1：プログラム・コード・サイズの増加率

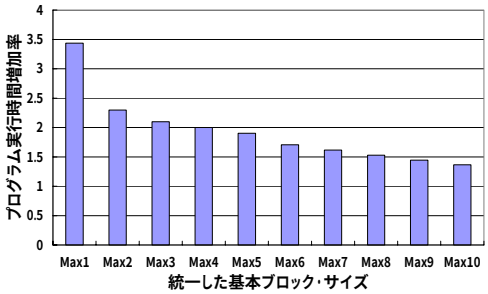


図 2：プログラム実行時間の増加率

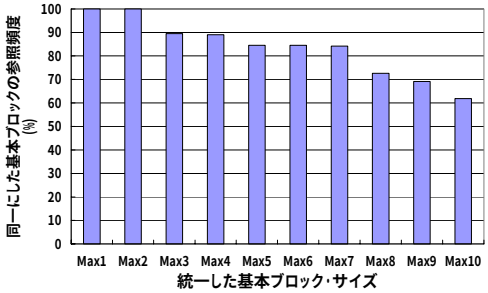


図 3：統一にした基本ブロックの参照頻度

## 4 おわりに

本稿では、基本ブロックサイズ統一に伴うコード・サイズ増加率、実行時間増加率、安全性の度合いの関係を評価した。今後の課題として、多くのベンチマーク・プログラムを用いてさらに評価を行う。


## 謝辞

日頃から御討論頂く、研究室の諸氏に感謝致します。


## 参考文献


[1]SimpleScalar/ARM URL: http://www.simplescalar.com/v4test.html

[2]SPEC(Standard Performance Evaluation Corporation) URL:http://www.spec.org/osg/cpu95/