様式Ｃ－１９

科学研究費助成事業（科学研究費補助金）研究成果報告書

平成２４年　５月　７日現在

機関番号：１４３０３
研究種目：基盤研究(C)
研究期間：2009～2011
課題番号：２１５００００５３
研究課題名（和文）　対不正攻撃セキュリティ機能を有する付加プロセッサに関する研究

研究課題名（英文）　A Coprocessor Architecture to Defend against Security Attacks

研究代表者
平田　博章（HIRATA　HIROAKI）
京都工芸繊維大学・工芸科学研究科・准教授
研究者番号：９０２７３５４９

研究成果の概要（和文）：　本研究では、バッファオーバフローの脆弱性を利用したスタックスマッシング攻撃に対する防御を目的として、攻撃の検出機構を備えたマイクロプロセッサアーキテクチャを提案した。プロセッサコアの外部に攻撃検出のための専用プロセッサを設け、その上で動作するソフトウェアを用いて攻撃検出を行うことで、攻撃の検出精度とプログラムの実行性能の両立を可能とした。

研究成果の概要（英文）：　We proposed a novel microprocessor architecture with a defense mechanism against stack smashing attacks, which corrupt the procedure return address and force the target computer to execute a malicious code. Our processor consists of two components; the main processor, which executes the operating system and user applications, and the auxiliary processor, which follows the program execution in the main processor and detects the return address corruption. Consequently, our architecture achieves both detection accuracy and performance.

交付決定額

（金額単位：円）

	直接経費	間接経費	合　計
２００９年度	500,000	150,000	650,000
２０１０年度	1,100,000	330,000	1,430,000
２０１１年度	800,000	240,000	1,040,000
年度			
年度			
総　計	2,400,000	720,000	3,120,000

研究分野：情報工学
科研費の分科・細目：情報学、計算機システム・ネットワーク
キーワード：計算機システム、システムオンチップ、インターネット高度化、
セキュア・ネットワーク

１．研究開始当初の背景

インターネットを介して不正アクセスを行う場合の有力な手段として、スタックスマッシング攻撃など、バッファオーバフローを意図的に発生させるものが知られており、重大な被害がもたらされてきた。それまで 10 年近くの間、その対策のために多くの研究が行われ、また、実際に多くの時間と労力を費やしてアプリケーションプログラムの修正が施されてきた。これによって被害件数は減少傾向にあるものの、決定的な解決策が見出されないまま、依然としてその脅威が取り除かれるには至らず、今日でもセキュリティパッチの配布と注意喚起が行われている現状にある。

２．研究の目的

バッファオーバフローの脆弱性を利用し

た攻撃には、スタックスマッシング攻撃などいくつかの手法があるが、その多くは、まず、リターンアドレスを不正に書き換えることを攻撃の手がかりとしている点で共通している。従って、リターンアドレスの書き換えを正確に検出することができれば、不正アクセスを防御することができる。そこで、本研究では、そのような防御機構を備えたプロセッサアーキテクチャを開発する。具体的には以下の技術を確立することで、バッファオーバフローの脆弱性を利用した攻撃に対する本質的かつ完全な解決策を提供することを目的とする。

(1) プログラムの関数（手続き）呼び出し状況を、自動的かつ正確に管理する方式を確立する。原則的には、関数の呼び出し関係はスタックを用いて容易に管理することが可能であるが、実用的なプログラムに対しても適用可能とするためには、C 言語における setjmp()/longjmp()を含むシステムライブラリの呼び出しや C++の例外処理（try～throw～catch）についても対応できなければならない。従来の提案ではこの点に問題があったが、本研究では実用性を重視してこれらの非局所分岐を例外視することなく扱う。

(2) アプリケーションプログラムの書き換えを行う必要なく、攻撃から防御可能な方式を開発する。

(3) 上記(1)に基づいて、実際に攻撃を受けた場合に正確にこれを検出し、かつ、攻撃を受けていないにも係わらず攻撃されたと誤検出しない方式を開発する。

(4) 上記(1)を実現するための管理情報を保存する方式を開発する。従来の提案では、ハードウェア資源の不足や割り込みの発生などによって管理情報を失ってしまい、その後は攻撃の検出漏れや誤検出が生じるなどの課題が残されていた。本研究では実用性を重視して、そのような課題を残さない正確な攻撃検出方式を、OS との関係および OS からの使いやすさを考慮して検討する。

(5) 上記(1)(3)(4)の機能を、アプリケーションプログラムの実行性能を低下させることなく実現するためのプロセッサアーキテクチャを提案する。

３．研究の方法

(1) 命令レベルにおいて、関数呼び出しと関数からの戻りを正確に検出し、それらをモニタリングすることで関数呼び出しの論理的な関係を把握する方式を検討する。これは ABI (Application Binary Interface)やコンパイラにも関連し、それらとの独立性を維持する必要がある。また、ライブラリ関数では、リロケーション情報を取得する目的で関数呼び出し命令を用いる場合があり、これをソースプログラムレベルでの関数呼び出しと区別できなければならない。さらに、実際の使用に耐え得る技術として提案を行うためには、C 言語における setjmp()/longjmp()や C++における例外処理に対応できなければならない。これらの機能は、一般的に、プログラム自らがリターンアドレスを書き換えることによって実現されているので、バッファオーバフローによるリターンアドレスの上書きと区別できなければならない。

(2) 下記(3)の検討を行うための予備評価として、アプリケーションプログラムの挙動評価を行う。一般的かつ実用的なアプリケーションプログラムを対象に、バッファオーバフローによって問題となり得る分岐命令からどのくらいの時間で攻撃検出を行わなければならないかを、マシン命令レベルの機能シミュレータを用いて調査を行う。

(3) 上記(1)～(2)の結果を基に、システムアーキテクチャの設計を行う。

(4) 上記(3)の結果を基に、性能評価のための時間シミュレータを開発する。このシミュレータは IBM PowerPC の命令セットを対象とし、実行サイクル単位で精緻なプログラム実行時間を計測する。

(5) 上記(4)で開発した時間シミュレータを用いて性能評価を行い、本研究の有効性を検証する。ここでの評価は、攻撃検出のために法外なオーバヘッドが生じないことを確認するのが第一の目的であり、更に、必要があれば方式上の改良を行う。

４．研究成果

(1) 関数呼び出し関係を正確に把握・管理するための論理的な機構として、図 1 に示す DRAS (Defensive Return Address Stack)機構を開発した。DRAS は図 1(b)に示すように、リターンアドレスを保存するための RAS(Return Address Stack)と、特に setjmp()からのリターンアドレスを保存するための SJS (Set Jump Stack) の 2 つのスタックから構成する。RAS の 1 つのエントリにはリターンアドレスのみを保存し、SJS の 1 つのエントリには、リターンアドレスのほかに、longjmp()の実行時にトップとすべき RAS のエントリの位置情報を保存する。

DRAS 操作のための専用命令として、

① オペランドで指定された値 i を用いて、RAS のトップから数えて i 番目のエント

リに保存されているsetjmp()からのリターンアドレスをSJSのトップエントリにコピーし、コピー先のRAS entryフィールドの値をRASのトップから数えてi+1番目のエントリを指すように設定する機能を持つdcl_setjmp命令、

② オペランドで指定されたリターンアドレスがRAS内に存在するか否かを，RASのトップから下方に向かって検索する機能を持つunwind_test命令、

③ 通常のリターン命令としての機能のほかに、以前に実行したunwind_test命令によって保存したunwind_targetの位置までRASのエントリをポップする機能を持つtrusted_return命令、

の3種の命令を新たに設けることで、非局所分岐に対応する。DRASはアプリケーションプログラムからは直接的には見えず、また、上記の専用命令は標準ライブラリ関数内で使用するための命令であり、したがって、アプリケーションプログラムの書き換えを行う必要なく、C言語における setjmp()/longjmp()やC++における例外処理を含む実用的なアプリケーションプログラムに対して正確な攻撃検出を行うことが可能となった。

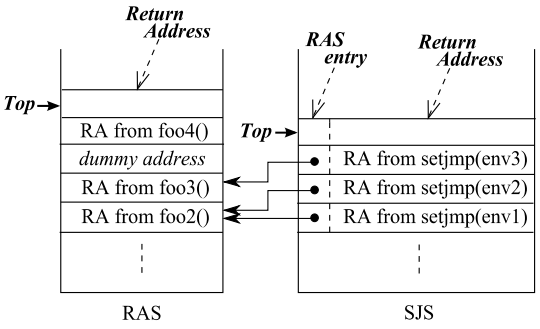
マシン命令セットにおいて、手続き呼び出しに使用される分岐命令をコール命令、手続きからのリターンに使用される命令をリターン命令とそれぞれ呼ぶことにする。コール命令の実行時には、単にRASのトップエントリにリターンアドレスをコピーする。実行したコール命令が setjmp()を呼び出すものである場合、後に setjmp()の中で実行されるdcl_setjmp命令により、リターンアドレスをさらに SJS のトップエントリにコピーし、longjmp()の実行時にトップとすべき RAS のエントリの位置情報を設定する。例として、図1(a) のプログラムにおいて関数foo4()を実行している時点でのDRASの内容を図1(b)に示す。

なお、システムライブラリ内の関数が自身のリロケーション情報を得るためにコール命令を使用する場合があるが、これはマシン命令レベルでは手続き呼び出しと区別がつかないため、通常のコール命令と同様にその情報をRASに保存する。図1の例では、関数foo3()内でこのような目的でコール命令を実行したものと想定して、これを dummycallと記述している。

リターン命令の実行時には、まずRASのトップから下方に向かって、リターン命令の分岐先アドレスとRASのすべてのエントリを比較する。比較が一致した時点で、RAS のトップからそのエントリまでをポップして処理を終了する。先に述べた特殊な目的でのコール命令が使用された場合にも正しく攻撃検

```
foo1() { … foo2() ; … }
foo2() { … setjmp(env1) ; setjmp(env2) ; foo3() ; … }
foo3() { … dummycall ; setjmp(env3) ; foo4() ; … }
foo4() { … }
```

(a) プログラム



(b) DRASの内容

図1　DRAMの構造（例）

出を行うために、RAS のトップだけでなく、その下方にあるエントリに対しても比較処理を行う。ただし、実際には、多くの状況においてRASのトップに一致するリターンアドレスが保存されており、最大でも2個のエントリに対して比較を行えば十分である。

上記の比較が一致しなかった場合、SJS のトップから下方に向かって、リターンアドレスが一致するエントリが見つかるまでSJS内を検索する。一致するエントリが見つかった場合、RAS のトップから、SJS の RAS entryフィールドが指す位置までのすべてのエントリをポップする。一致するエントリが見つからなかった場合、その分岐先アドレスは攻撃によって改竄されたものであるとして攻撃検出信号を生成する。

オブジェクト指向言語における例外発生時においては、処理系で用意された関数の中で実行されるunwind_test命令により、スタックフレーム上のリターンアドレスの改竄の有無を検査すると同時に、RAS をどこまでポップすべきかを記憶する。これにより、trusted_return命令を用いてcatch節へ分岐する際に、RAS のトップを正しく更新することができる。なお、trusted_return命令のオペランドに指定する分岐先アドレスは、ライブラリ関数内でcatch節の先頭アドレスに自ら書き替えたものであり、改竄の有無を検査する必要はない。

(2) プログラムの実行性能の低下を抑えるためには、DRASをハードウェアで実装することが望ましい。しかし、DRASに対する操作は、コール命令、リターン命令、非局所分岐に対応するための専用命令、のそれぞれで異なり、これらに対する処理をハードウェアのみで実現するのは得策とはいえない。また、DRASを構成する2つのスタックの容量が不足する場合やコンテキストスイッチが発生した際に、DRAS内の情報を失わないためにメモリ上

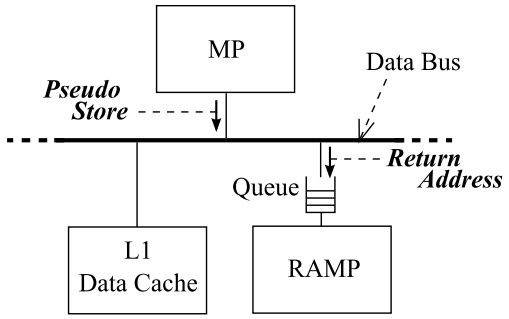


図３　システム構成

に退避するなどの対策が必要となる。これはプログラム実行の性能低下を招くだけでなく、OS に対しても少なからぬ変更を必要とする。

そこで本方式では, OS やユーザプログラムを実行する通常のプロセッサコアの外部に、DRAS に関する処理を行う専用のプロセッサを設け、その上で実行する組込みソフトウェアによって DRAS の機能を実現する。以後、通常のプロセッサコアをメインプロセッサ（MP; Main Processor）、DRAS 操作を担当する専用のプロセッサをリターンアドレス管理プロセッサ（RAMP; Return Address Management Processor）、とそれぞれ呼ぶことにする。

RAMP 上でのソフトウェアを用いた柔軟な処理により、ハードウェアのみでは困難な DRAS 操作を行うことが可能となる。また、MP の外部でリターンアドレスを独立して管理するため、コンテキストスイッチの発生時にスタックのエントリをメモリ上に退避する必要はない。さらに、例えば、メインメモリの数ページを固定的に RAMP の占有領域として割り当て、RAMP 上のソフトウェアで DRAS 操作を行うことにより、DRAS のサイズについても柔軟に制御することが可能となる。

RAMP は DRAS 操作とそれに関わる攻撃検出処理を行うための必要最低限の機能を持った小規模なプロセッサコアで構成する。RAMP 上で実行するプログラムは DRAS 操作に特化した単純なものであり、MP 上の OS から一切のランタイムサービスを受ける必要がない。したがって、MP 上の OS は RAMP を単なるハードウェア機構として扱うことができる。このようなシステム構成上の観点からは、RAMP 上のソフトウェアには、MP 上の OS で扱うコンテキスト ID に関連づけて DRAS 操作を行う機能が必要となる。

MP と RAMP を機能分担して構成したことにより、MP がコール命令やリターン命令を実行した際にリターンアドレスを RAMP に転送する必要が生じる。しかし、リターンアドレスの送信経路として、MP の命令デコーダから RAMP へ専用のデータパスを設けることは、配線及びピン制約の観点から実現が困難である。そこで、図２に示すように、MP とデータキャッシュとを繋ぐデータバスに RAMP を接続することにより、実装に要するハードウェア量の増大を回避した。

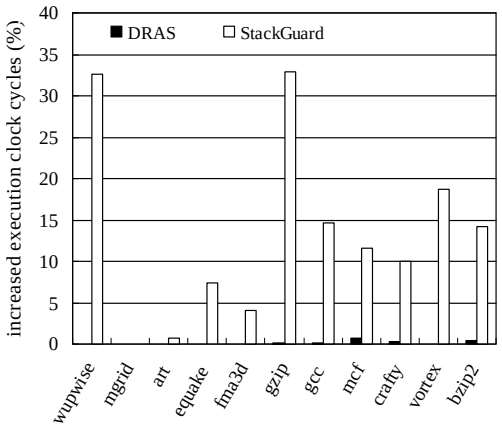


図２　実行サイクルの増加率

(3) MP はコール命令やリターン命令を実行する際、命令デコーダが生成した内部オペレーションを、ロードストアユニットを経由して RAMP へ転送する。そのため、MP が実行したリターン命令に対して RAMP が攻撃検出処理を開始するまでに時間的な遅延が生じる。しかし、実際には、たとえ攻撃を受けていたとしても、リターン命令の実行時点で即座に攻撃検出を完了する必要はない。プログラムがシステムコールを行うごとに、それまでに攻撃を受けていないことが確認できれば、攻撃の影響をそのプログラムのみに封じることができ、システム全体に被害が及ぶことはない。逆に言えば、それまでに実行されたすべてのリターン命令に対する攻撃検出の処理結果が得られるまで、MP においてシステムコールの実行を遅らせる必要がある。そこで、MP 内に専用のカウンタを用意し、システムコールの実行を制御する。MP がリターンアドレスを RAMP へ送信するごとにカウンタの値を 1 増加させ、RAMP から処理結果を得るごとにカウンタの値を 1 減少させる。MP はシステムコールを実行する際にこのカウンタを参照し、その値が 0 になるまではシステムコールを実行しない。

(4) SPEC CPU 2000 ベンチマークの中の 11 種類を用いて各ベンチマークプログラムの実行サイクル数をシミュレーションにより測定し、DRAS を適用した場合のプログラムの実行サイクル数の増加率を計測した。MP は 4 ウェイのスーパスカラプロセッサであり、一方、PAMP は 1 サイクル当たり 1 命令発行の単純なパイプラインプロセッサで構成した。また、比較のため、既存の防御方式のひとつである StackGuard を適用したプログラムを MP で実行した場合についても同様の計測を行った。

計測した実行サイクル数の増加率を図３に示す。計測したすべてのプログラムにおいて、DRAS を適用した場合の実行サイクル数の増加率は0.7%以下であった。一方で StackGuard を適用した場合は、プログラムによって値が大きく異なるが、全体として DRAS よりもはるかに高い値を示しており、gzip では 32.9%の性能低下をもたらす。両方式の比較からも、DRAS がプログラムの実行性能に与える影響が極めて小さいことが確認できた。

(5) さらに、1 基の RAMP で同時に 2 基の MP の攻撃検出を行う場合の性能評価を行い、この場合もほとんど性能低下をもたらすことなくアプリケーションプログラムを実行することができることを確認した。これにより、マルチプロセッサ構成のコンピュータにも本方式が適用可能であることを示した。

５．主な発表論文等
（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計８件）

① 中務国男，山田徹，布目淳，平田博章，柴山潔，スタックスマッシング攻撃の正確な検出機構を備えたプロセッサアーキテクチャ，第 10 回情報科学技術フォーラム論文集，査読有，Vol.1，pp.63-66，2011.
② 藤井崇弘，森田清隆，布目淳，平田博章，柴山潔，スレッドレベル並列化のためのメモリリネーミング，第 10 回情報科学技術フォーラム論文集，査読無，Vol.1，pp.385-388，2011.
③ 赤坂謙二郎，布目淳，平田博章，柴山潔，電子商取引サイトにおける応答待ち時間の短縮を目的としたデータベース参照のスケジューリング方式，第 10 回情報科学技術フォーラム論文集，査読有，Vol.1，pp.113-116，2011.
④ 布目淳，平田博章，柴山潔，IPv6 環境におけるネットワーク認証のためのマルチキャストフィルタリングイーサネットスイッチ，第 10 回情報科学技術フォーラム論文集，査読有，Vol.4，pp.1-4，2011.
⑤ A.Nunome，H.Hirata，M.Fukuzawa，and K.Shibayama，Development of an E-learning Back-end System for Code Assessment in Elementary Programming Practice，SIGUCCS Fall 2010 Conference，ACM，査読有，pp.181-186，2010.
⑥ 平田博章，山田徹，布目淳，柴山潔，マシン命令レベルでのプログラム実行モニタリングによる手続き呼び出し関係の正確な検知方式，第９回情報科学技術フォーラム論文集，査読有，Vol.1，pp.87-90，2010.
⑦ 森田清隆，布目淳，平田博章，柴山潔，スレッドレベル並列化のためのスレッド間依存関係の分類，第９回情報科学技術フォーラム論文集，査読有，Vol.1，pp.81-86，2010.
⑧ 赤坂謙二郎，布目淳，平田博章，柴山潔，データベース参照のスケジューリングによる電子商取引サイトの最適化，第 9 回情報科学技術フォーラム論文集，査読無，Vol.1，pp.407-408，2010.

〔学会発表〕（計４件）

① 藤井崇弘，森田清隆，布目淳，平田博章，柴山潔，スレッドレベル並列投機実行のためのメモリリネーミング機構，電子情報通信学会 2012 年総合大会 学生ポスターセッション，於・岡山大学，2012 年 3 月 22 日.
② 安井寛幸，布目淳，平田博章，柴山潔，基本ブロック単位の命令実行パス予測方式，電子情報通信学会 2012 年総合大会 学生ポスターセッション，於・岡山大学，2012 年 3 月 22 日.
③ 野間翔平，布目淳，平田博章，柴山潔．スタックスマッシング攻撃の正確な検出方式とその性能制約条件，電子情報通信学会技術研究報告 CPSY2009-47，Vol.109，No.319，pp.25-30，於・高知市文化プラザ，2009 年 12 月 4 日.
④ 野間翔平，布目淳，平田博章，柴山潔，バッファオーバフロー検知用付加プロセッサの概要，情報処理学会第 71 回全国大会論文集，Vol.1，pp.41-42，於・立命館大学，2009 年 3 月 11 日.

６．研究組織

(1)研究代表者
平田　博章（HIRATA　HIROAKI）
京都工芸繊維大学・工芸科学研究科・
准教授
研究者番号：９０２７３５４９

(2)研究分担者
柴山　潔（SHIBAYAMA　KIYOSHI）
京都工芸繊維大学・工芸科学研究科・教授
研究者番号：７０１２７０９１

(3)連携研究者
（　　）

研究者番号：