バッファオーバフロー検知用付加プロセッサの概要

野間 翔平 布目 淳 平田 博章 柴山 潔
京都工芸繊維大学大学院 工芸科学研究科

1. はじめに

近年，バッファオーバフローを利用した不正アクセスが問題になっている．この攻撃を受けると，システムの制御が奪われるなどの深刻な被害をもたらす．その対策として，文字列の長さを厳密に管理する専用ライブラリを用いたり，バッファオーバフローを検知するためのチェックコードを挿入するなどの防御方式が提案されている．

しかし，これらの方式ではプログラムの実行性能の低下が問題となる．例として，上記の専用ライブラリの1つである Safestr ライブラリ[1]が，通常の C 標準関数に対してどの程度の実行時間を要するかを実測により評価し，その結果を表 1 に示す．処理する文字列の長さにもよるが，Safestr ライブラリの関数が C 標準関数に比べて数百倍の処理時間を要することがわかる．

本稿では，プログラマに対してバッファオーバフロー対策に関する負担を強いることなく，また，プログラムの実行性能の低下を招かずに，バッファオーバフローを検知する新たな手法を提案する．

2. バッファオーバフローの概要

ユーザプログラムの実行時には，関数内におけるローカル変数の値や戻り番地などの情報は，メモリ上に割り付けられたスタックフレームに保存される．図 1 にスタックフレームの例を示す．1個のスタックフレームはフレームポインタとスタックポインタ（Stack Pointer：以下 SP とする）の組で指定される．

バッファオーバフローは，このスタックフレーム内の変数操作時などに発生する．例えば，図 1 の関数 func 内で strcpy 関数などの文字列処理を行い，*str*[5]で確保された領域に長さが 5 以上の文字列を格納しようとすると，バッファオーバフローが発生する．このとき，スタックフレーム内に保存していた関数の戻り番地までもが書き換えられると，関数 func から関数 main へ戻る時点で，書き換えられた番地のコードを実行させることが可能となる．

攻撃の典型例は，攻撃者が悪意あるコードを含むメッセージを送信してバッファオーバフローを発生させ，戻り番地をそのコードの開始番地に改ざんするものである．

3. バッファオーバフロー検知方式

本稿では，実行中のユーザプログラムをマシン命令レベルで監視し，関数からリターンするたびにその戻り番地の正否をリアルタイムにチェックすることで，バッファオーバフローの発生を検知する方式を提案する．

表 1 Safestr ライブラリの標準関数に対する処理時間比

関数名	処理時間比
sprintf	460
strcmp	155
strcpy	195

＊処理時間比＝
Safestr ライブラリの処理時間 / 標準関数の処理時間
＊計測マシン
CPU：PowerPC G5 (2.5GHz)

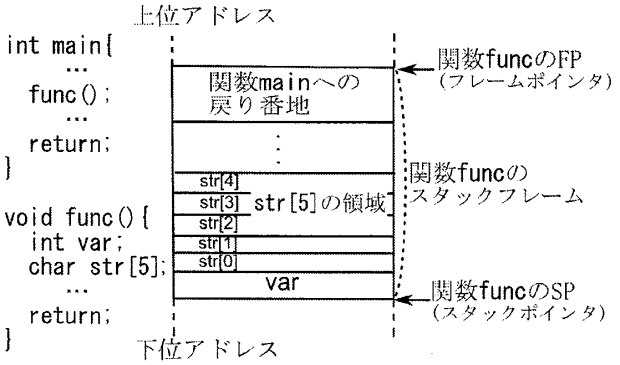

図 1 スタックフレームの概要

ユーザプログラムで関数呼び出しが行なわれるたびに，その戻り番地をスタックフレームとは別の領域にコピーし，これを LIFO 構造で管理する．このデータ構造を**コールスタック**とよぶことにする．コールスタックのトップエントリには現在実行中の関数の正しい戻り番地を記憶しているので，リターン時に戻り番地（改ざんされている可能性がある）とコールスタック内の戻り番地とを比較することによって，バッファオーバフローの発生を検知する．

3.1 関数呼び出しの検出

実行中のプログラムを監視し，コールスタックを用いて戻り番地を管理するためには，プログラミング言語レベルでの関数呼び出しとリターンを，マシン命令レベルで正確に検出する必要がある．本稿で対象とする PowerPC アーキテクチャ（以下 PPC とする）では，プログラミング言語レベルでの関数呼び出しとリターンは，分岐命令（PPC では b, bc, bcctr, bclr の 4 種類がある）によって実現されているが，分岐命令は関数呼び出しとリターン以外の目的でも用いられるため，単純に分岐命令を検出するだけでは関数呼び出しとリターンを検出できない．

本方式では，分岐命令の LK フィールドの値を参照することで，その分岐命令がどのような目的で使用されるかを判定する．LK フィールドの値が 1 の場合には，分岐命令実行時にその分岐命令の次の番地（戻り番地）をレジスタ LR に格納する．表 2 に，プログラミング言語における実行制御機能と，それを実現する分岐命令およびその LK フィールドとの対応をまとめる．

表 2 より，LK フィールドの値が 1 の分岐命令（以下，これらを**コール命令**とする）の実行を検出して，これを関数呼び出しと判定する．また，PPC のアプリケーションバイナリインタフェース[2]により，関数からのリターンには bclr 命令（レジスタ LR に格納された番地へと分岐を行う命令）を使用することが定められているので，bclr 命令（以下，これを**リターン命令**とする）の実行を検出してリターンであると判定する．

3.2 特殊な関数などに対する処理

戻り番地をコールスタックのトップエントリに記憶した

An Attached Processor for Buffer-Overflow Detection

Shohei NOMA, Atsushi NUNOME, Hiroaki HIRATA, Kiyoshi SHIBAYAMA
Graduate School of Information Science, Kyoto Institute of Technology

表 2 条件別分岐命令の実装方法

プログラミング言語における実行制御機能	分岐命令の LK の値			
	b	bc	bcctr	bclr
静的な関数の呼び出し	1	1	-	-
動的な関数の呼び出し（関数ポインタ使用）	-	-	1	-
関数からのリターン	-	-	-	0
その他（条件式，ループなど）	0	0	0	-

＊ 表中の - はその分岐命令が使用されないことを示す

値と単純に比較するだけでは，以下に挙げる特殊な場合に，正常なプログラムの実行状態であってもバッファオーバフローが発生したものと誤検知してしまう．

- longjmp 関数
- リロケーション情報の取得
- C++などにおける例外処理

これらの場合に対処するため以下の機能や命令を設ける．

- dcl_setjmp 命令（setjmp 関数の実行を宣言する）
- コールスタック内での SP の管理機能
- bctlr 命令（戻り番地のチェックを行わないことを明示する bclr 命令）

3.2.1 longjmp 関数を考慮した処理手順

まず，コールスタックには，関数呼び出しを行なった時点の SP の値を，戻り番地と組にして記憶する．また，ユーザプログラムにおいては，setjmp 関数内で必ず dcl_setjmp 命令を実行するものとする．この dcl_setjmp 命令の実行を検出して，コールスタックのトップエントリのコピーをコールスタック内に作成し，将来の longjmp 関数の実行に備える．ユーザプログラムでは longjmp 関数の実行によって，先の setjmp 関数の戻り番地へと制御が移るが，この処理は一般的には longjmp 関数内で戻り番地を変更した後に，通常のリターン命令を実行することによって実現されている（ただし，その戻り番地が改ざんされている可能性もある）．

以上を考慮して，リターン時のバッファオーバフローの検知は以下の手順で行う．

1. リターン命令を検出すると，リターン命令実行時点での SP の値と一致する SP を持つエントリを，コールスタックのトップから順に検索する．
2. SP の比較が一致したエントリに対して，その戻り番地とリターン命令の分岐先とを比較する．両者が一致した場合は，正常なリターンであると判定し，検知処理を終了する．一致しない場合は 3 へ．
3. さらに，リターン命令実行時の SP と同じ値を持つエントリを検索し，そのようなエントリが存在すれば 2 へ．存在しない（ボトムまで検索が終了した）場合は，バッファオーバフローが発生したと判定し，検知処理を終了する．

dcl_setjmp 命令によってコピーしたエントリとそれ以外のエントリとは区別して扱う必要があるが，その詳細については本稿では省略する．

3.2.2 関数呼び出し以外のコール命令への対応

ライブラリ関数の中には，リロケーション情報を取得する手段としてコール命令を使用する場合がある．これは実際には関数呼び出しではないが，マシン命令レベルで判別することは困難である．

本方式では，そのような目的でコール命令が実行された場合も，関数呼び出しとしてコールスタックに情報を追加する．しかし，リターン時には 3.2.1 で述べたように SP の比較を行いながら戻り番地をチェックするので，バッファオーバフローを誤検知することはない．

3.2.3 C++における例外処理への対応

例外処理が行われる場合，もしもスタックフレーム中の関数本来の戻り番地がバッファオーバフローによって改ざんされていたとしても，例外処理用のライブラリルーチンによって，戻り番地が catch 節の開始番地に更に書き換えられる．このため，バッファオーバフローによる攻撃はその目的を達成できない．

しかし，catch 節の開始番地をコールスタックを用いて認識することは不可能である．そこで，これらの処理が行われる場合，bctlr 命令を用いて戻り番地のチェックを省略することで，バッファオーバフローの誤検知を防ぐ．

3.3 防御方式の実装

本方式の実装については，監視という性質とリアルタイム検知の必要性から，ハードウェアで実現するのが望ましい．しかし，コールスタックの操作やバッファオーバフロー検知には 3.2.1 で述べた柔軟な処理が必要であることから，ソフトウェアによる実装も積極的に取り入れる．

結論として，ユーザプログラムや OS を実行する通常のプロセッサ（以下メインプロセッサとする）に対して，本方式のバッファオーバフロー検知に必要最低限の機能を持った小規模な専用プロセッサ（以下**付加プロセッサ**とする）を設ける．この付加プロセッサはメインプロセッサにおけるマシン命令の実行を監視し，バッファオーバフローを検知した場合は割り込みによってメインプロセッサに通知する．

この割り込みはリターン命令を検出した直後に通知するのが望ましいが，3.2.1 で述べた検索処理を行う必要があるため，リターン命令の検出後に直ちに検知処理を完了できるとは限らない．したがって，バッファオーバフローによる攻撃を受けた場合，悪意あるコードが実行される可能性がある．しかし，実際には，メインプロセッサにおいてシステムコールが発行されるまでは，悪意あるコードが実行されたとしても，実行中のプログラムが一時的に破壊されるだけで，被害がシステムに及ぶことはない．よって，付加プロセッサによる割り込みの通知は，メインプロセッサにおける次のシステムコールまでに行えればよい．

付加プロセッサはメインプロセッサと並列に動作するので，プログラムの実行性能を低下させることなく，バッファオーバフローによる攻撃から防御することが可能となる．

4. まとめ

本稿では，ユーザプログラムの実行時における戻り番地をリアルタイムに監視することで，バッファオーバフロー攻撃に対して実用性を備えた防御方式を提案した．

今後は，本方式の実装について詳細な設計を行うとともに，シミュレーションによる性能評価を行う．

参 考 文 献

[1] Matt Messier and John Viega: “Safe C String Library”, http://www.zork.org/safestr/

[2] IBM Corp.: “Developing PowerPC Embedded Application Binary Interface (EABI) Compliant Programs”, PowerPC Embedded Processors Application Note, 1998.