C-004

キャッシュ上のタグビットを用いたバッファオーバーフロー防止手法

上田　和哉†　　小柳　滋†

†立命館大学大学院理工学研究科

1　はじめに

バッファオーバーフローの脆弱性をついた不正プログラムは，依然猛威を振るっている．この脆弱性を利用すると，システムの実行制御を乗っ取り，ユーザが意図していない任意のコードを実行することができる．ソフトウェアの改良により，脆弱性を修正することは可能だが，未知の脆弱性に対応することができない．したがって，バッファオーバーフローそのものを起こさないための手法が必要である．

バッファオーバーフロー防止手法にはソフトウェアで行うものとハードウェアで行うものがあるが，互換性を保つことができるという点と，未知の脆弱性に対応可能という点から，ハードウェアによる手法を採用する．本稿では，キャッシュの拡張によるバッファオーバーフロー防止手法を提案する．

2　バッファオーバーフローを用いた不正プログラムの実行

2.1　バッファオーバーフローとは

バッファオーバーフローとは，プログラムの変数用に用意された領域を超えて，データが書き込まれることである．C言語の標準ライブラリには，データを書き込むときに領域チェックを行わないものがあり，それがバッファオーバーフローの原因となっている．バッファオーバーフローには，スタックオーバーフローとヒープオーバーフローがあるが，本稿ではスタックオーバーフローを対象とする．

2.2　スタックスマッシング

通常，スタックは上位アドレスから下位アドレスに成長し，データ領域は下位アドレスから上位アドレスに向かって成長する．このため，データ書き込みのときにバッファオーバーフローが起こると，スタック領域のデータが上書きされてしまう．このことをスタックオーバーフローという．スタックオーバーフローが発生すると，スタックに保存されているリターンアドレスが書き換えられて，任意のコードが実行される．攻撃者はこの性質を利用し，リターンアドレスを不正プログラムの開始アドレスに書き換えることで，コンピュータへの攻撃を行う．この攻撃はスタックスマッシングと呼ばれている．

スタックスマッシングの例を図１に示す．ここでは関数 f によって関数 g が呼び出され，g の関数中で strcpy によって文字列 s1 をコピーするという処理を考える．strcpy は文字列を書き込むときにデータの境界チェックを行わないので，バッファオーバーフローを起こす可能性がある．通常は，関数 g が呼び出された時に関数 f へのリターンアドレスがスタックに保存され，関数 g の処理終了後，スタックからリターンアドレスを読み出してプログラムカウンタの値を復元し，関数 f の処理に戻る．しかし，文字列 s1 のサイズが関数 g で確保した buf の大きさを超えていると，buf の領域を超えてデータが書き込まれてしまう．文字列 s1 が buf より大きく，改ざんしたリターンアドレスと悪意あるコードを含んでいる場合，strcpy によって悪意あるコードがスタック内に書き込まれ，リターンアドレスが悪意あるコードの開始アドレスに書き換えられてしまう．したがって，関数 g の処理が終了したあとプログラムは関数 f の処理には戻らず，悪意あるコードを実行することになる．

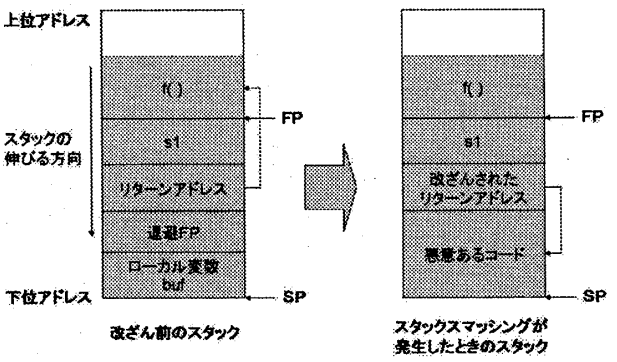

図 1: スタックスマッシングの例

3　提案手法

本手法では，キャッシュのタグメモリに書き込み禁止を示すためのタグビットを付加し，手続き情報領域への上書きを禁止することで，バッファオーバーフローを防止する．この書き込み禁止の判定は，ライン単位で行う．具体的な動作は次のとおりである．

- 手続き情報を扱う場合
  手続き呼び出しを行うとき，リターンアドレスやフレームポインタなどの手続き情報を保存し，そのラインのタグビットの値を１にする．当該情報を読み出した時は，そのラインのタグビットの値を０にする．
- その他のデータを書込む場合
  対象のラインにアクセスした時に，タグビットの値を調べる．タグビットの値が１の場合，書き込みアクセスを中止し，プロセッサに例外の発生を通知する．プロセッサは例外の通知を受け，そのプログラムの実行を停止する．

提案手法は，キャッシュコントローラを拡張することにより，ハードウェア処理で解決するため，プログラムコードの互換性を保つことができる．ただし，手続き呼び出しを行う際にはタグビットを操作するために，手続き情報を保存するということをキャッシュに通知する必要がある．そのために，発行された命令が手続き情報を扱うものであることを通知するように，プロセッサの仕様を変更する．書込み時のタグビットの比較はキャッシュコントローラで行うため，他の部分の変更は必要ない．また，書き込み中止はプロセッサの例外処理を使うため，これも変更する必要はない．したがって，提案手法を実装するにあたって，キャッシュ以外の部分は大きな変更をせずにすむ．

本手法ではタグメモリにビットを付加することにより手続き情報を保存するので，当該データがキャッシュから追い出されてしまうと保護できなくなってしまう．したがって，安全性を高めるために，タグビットが１となっているラインは

Protecting Buffer Overflow by using Tag Bit on Cache
†Kazuya UEDA　†Shigeru OYANAGI
†Graduate School of Science and Engineering,Ritsumeikan University

ミスヒットによる置き換えをしないものとする．このことにより，手続き情報が追い出されることはなくなるが，そのラインに対して別の手続き情報の書き込みを行うときは衝突が起こることがある．この場合，空いているウェイに書き込むことになるが，それでも手続き情報が衝突した場合，その書き込み先アドレスを書き込み禁止アドレスリストとして，一時的に保存することとする．そのためのバッファは別途設けることとする．このときの動作は次のようになる．

- 手続き情報を扱う場合
  手続き情報を書き込むとき，衝突が起こった場合，別のウェイに書込む．もしキャッシュに空きがなければ，その書き込み先アドレスをバッファに保存する．バッファ内に保存されているアドレスと一致するアドレスのデータが読み出されたとき，当該アドレスをバッファから削除する．
- その他のデータを書込む場合
  データにアクセスするとき，バッファにアドレスが入っている場合は，タグビットの参照と同時に，バッファ内も調べる．バッファ内のアドレスと書き込み先アドレスが一致する，あるいはタグビットが 1 であった場合，書き込みは中止される．

この場合，タグの比較とバッファ内のデータの比較を同時に行うので，バッファには連想メモリを用いる必要がある．

図 2 に，このキャッシュのブロック図を示す．

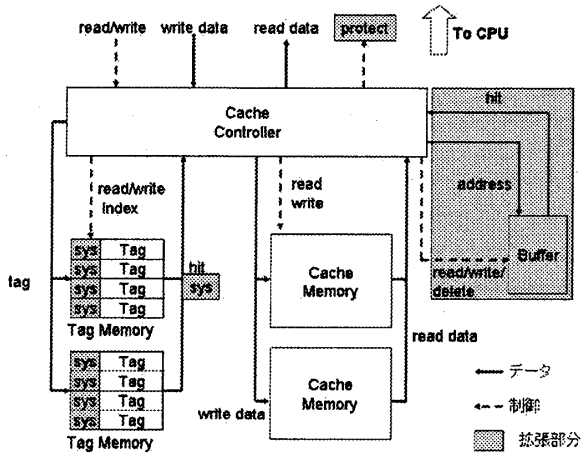

図 2: 拡張キャッシュモデル

タグメモリを 1 ビット分拡張し，アドレスバッファを追加している．

4 関連研究

ハードウェアによるスタックスマッシング攻撃の防止手法はいくつかある．いずれもスタックスマッシング攻撃を防ぐことを目的としているが，バッファオーバーフローを検出して実行を停止するものとバッファオーバーフローそのものを防ぐものがある．

John P. McGregor らの提案した SRAS[1] では，手続きを呼び出すときにリターンアドレスを通常のメモリとは別のハードウェアスタックに保存する．その後，リターン命令が発行されたときに返されたリターンアドレスと保存したリターンアドレスを比較し，異なっていればスタックスマッシングが起きたと判定する．この方法は，LIFO と異なる動作に対応できないという問題がある．

井上が提案する SCache[2] は，手続き情報の含まれたキャッシュラインのレプリカを複数作成し，リターン命令の実行時に，返されたリターンアドレスとレプリカラインのリターンアドレスを比較し，スタックスマッシングの発生を検出する．提案手法では，追い出しを禁止することにより，より安全性を高めている．

蛭田らの提案した手法 [3] では，スタックを手続き情報領域とローカル変数領域に分け，その境目をリミットアドレスとして保存する．書き込み時に，書き込み先アドレスがどの領域に属しているかをリミットアドレスと比較することによって調べ，手続き情報領域への書き込みを制限することでスタックスマッシングを防止する．書き込み命令を実行するときのオーバーヘッドが問題なので，提案手法ではその改善を図っている．

5 評価

以上の項目について，評価を行う予定である．

- 性能オーバーヘッド
  提案手法は，キャッシュからのデータの追い出しを禁止する場合がある．このとき使用できるキャッシュ容量が減るので，その分ヒット率が下がり，性能低下が起こることが考えられる．提案手法は安全性を重視しているが，性能オーバーヘッドが大きすぎると問題なので，性能と安全性のトレードオフを図る必要がある．この評価は，SimpleScalar ツールセットを用いて提案手法を実装し，ベンチマークプログラムによって実行速度を比較することによって行う．
- 安全性
  バッファを用いたとしても，ハードウェアの容量には限界がある．したがって，手続き情報をどの程度保護できるのかを検証する必要がある．それに伴って，安全性を保つために必要なキャッシュ容量及びアドレスバッファの適切な容量を求める．この評価は，SimpleScalar ツールセットを用いて実行プログラムをトレースし，衝突などがどのくらいの頻度で起きているかを調べる．
- ハードウェア増加量
  提案手法を実装する際，タグビット用のメモリ，アドレスバッファ，キャッシュコントローラの拡張により，ハードウェア量が増加する．したがって，提案手法の実装にはどの程度のハードウェアの追加が必要になるのかを見積もる．この評価は，XilinxISE ツールセットを用いて，キャッシュモデルを作成することによって行う．

6 おわりに

本稿では，キャッシュに付加したタグビットを用いたバッファオーバーフロー防止手法を提案した．今後シミュレーションを行い，性能オーバーヘッド，安全性及びハードウェア増加量を評価する．

参考文献

[1] John P. McGregor, David K. Karig, Zhijie Shi, and Ruby B. Lee. A processor architecture defense against buffer overflow attacks. *IEEE Internatiional Conference on Information Technology: Research and Education*, 2003.

[2] 井上弘士. 不正プログラムの実行防止を目的とするオンチップ・キャッシュ・アーキテクチャ. 情報処理学会研究報告 ARC-159, 2004.

[3] 蛭田智則, 山名早人. スタックフレームのセグメント化によるバッファオーバーフロー対策法. 情報処理学会研究報告 SLDM-119(28), 2005.