6W-03

例外処理機構を用いてStack-based Buffer Overflow攻撃を緩和するプログラムローダの提案と実装

金 雨淵† 近藤 秀太‡ 渡辺 亮平‡ 菅原 捷汰‡ 横山 雅展‡ 中村 慈愛† 齋藤 孝道†
明治大学† 明治大学大学院‡

1 はじめに

メモリ破壊脆弱性の一つにStack-Based Buffer Overflow(以下，SBoFという) 脆弱性（CWE-121) [1] がある．この脆弱性を悪用するSBoF攻撃は，プログラムの流れを任意に変更し，管理者権限の奪取を引き起こす恐れがある．これまでに，SBoF攻撃を防ぐための様々な対策技術が考案されてきた．先行研究 [2] では，アプリケーションとしてのプログラムローダ（以下，Safe Transローダという）を用いてSBoFを招くライブラリ関数を境界検査処理を追加した安全な関数に置換する手法を提案した．本論文では，Safe Transローダにおける境界検査処理を改善し，その評価を行った．Safe Transローダは，32ビットLinux OSにおけるELF形式の実行バイナリへの適用を想定する．

2 提案方式

2.1 Safe Transローダの概要

Safe Transローダは，アプリケーションとして動作するプログラムローダであり，OSの標準ローダに代わって，コマンドライン引数として指定された実行バイナリをロードする．その際，我々が選定した23個のSBoF脆弱性を招くライブラリ関数を，境界検査処理を追加した安全な関数（以下，境界検査関数という）に置換することで，その関数を起点としたSBoF攻撃を緩和する．Safe Transローダは，コンパイラによる対策と異なり，既に配布された実行バイナリにも適用できる長所がある．

2.2 境界検査関数

Safe Transローダの境界検査関数の仕組みについて説明する．境界検査関数は，Safe Transローダ内で定義されており，strcpy関数などのSBoF脆弱性を招くライブラリ関数とは異なり，書き込み先のバッファに文字列を書き込む前に，そのバッファの書き込み可能な上限サイズ（以下，上限サイズという）を算出し，入力された文字列長が上限サイズを超えていないかをチェックする境界検査処理が追加されている．SBoF脆弱性を招くライブラリ関数を悪用した攻撃では，境界検査処理がないことで図1(a)のように，スタックの高位にあるリターンアドレスが上書きされる恐れがある．一方で，Safe Transローダを適用する場合，境界検査処理により，図1(b)のように，リターンアドレスの上書きを防ぐことができる．

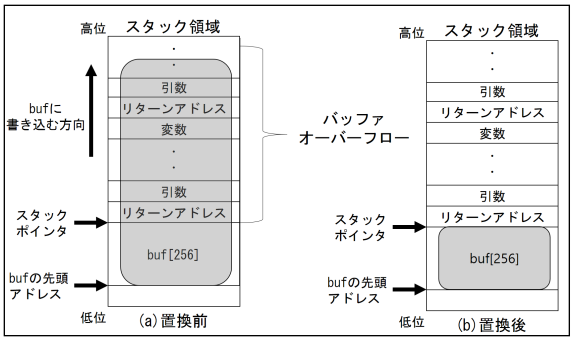

図 1: 提案方式の適用前後のスタック領域

先行研究の境界検査処理は，書き込み先のバッファの上限サイズを計算するために，フレームポインタを用いてスタックフレームを遡り，書き込み先のバッファが属する関数のスタックフレームを見つける実装となっていた．しかし，コンパイラの最適化オプションや，フレームポインタに関するオプション（-fomit-frame-pointer）により，フレームポインタを使用しないスタックフレームが存在する場合，正確にスタックフレームを遡ることができない問題があった．

本論文では，実行ファイル中の.eh_frameセクションの情報を用いてこの問題に対処した．ELFにおいて，関数ごとのスタックフレームの情報は，.eh_frameセクションに格納されている．スタックフレームの情報とは，実行ファイル内の.textセクションに含まれる複数の命令を一つの単位として，それらの実行時におけるスタックやレジスタの状態を指す．これらはデバッグ用のデータ形式であるDWARFに従って記述されている．通常.eh_frameセクションは，C++における try-catch 構文のような例外処理に用いられ，このセクション内の情報に従ってスタックやレジスタの状態を復元することで，スタックフレームを遡り throw された例外の型に対応

Proposal and Implementation of a Program Loader to Mitigate Stack-Based Buffer Overflow Attack with Exception Handling
†Kim Wooyeon ‡Kondo Shuta ‡Watanabe Ryohei ‡Sugawara Shota ‡Yokoyama Masahiro †Nakamura Jiai †Saito Takamichi
†Meiji University ‡Graduate School of Meiji University

する catch ブロックを探索する．本論文では，.eh_frame セクションの情報を扱うために libunwind [3] というライブラリを用いて境界検査処理を実装した．

改善した境界検査関数の動作について説明する．はじめに，libunwind の unw_step 関数を実行することで，スタックフレームを遡っていく．それに伴い unw_get_reg 関数を実行し，遡ったスタックフレームの基底アドレス（以下，スタックポインタという）を取得する．取得したスタックポインタが書き込み先のバッファの先頭アドレスよりも高位になった時点で，そのスタックポインタは，書き込み先のバッファを含むスタックフレームを指すことになる．ここで，unw_step 関数の実行を打ち切り，書き込み先のバッファの上限サイズの計算に移る．この時点でのスタックポインタは，当該スタックフレーム内のリターンアドレスの 4 バイト高位を指している．そこで，その値から 4 バイトを引くことで，リターンアドレスの位置を取得し，書き込み先のバッファの先頭アドレスとの差を上限サイズとして算出する．その後は，入力された文字列長が上限サイズを超えていないかチェックし，超えていない場合は書き込み処理を実行する．超えている場合，その旨を標準エラーに出力してプログラムの実行を中止する．

3 評価

3.1 有効性の評価

CWE-121 で公開されているサンプルコード Example 1，CVE-2013-4256 [4] が報告されている Network Audio System1.9.3 および CVE-2017-14493 [5] が報告されている dnsmasq2.70 を用いて，改善後の Safe Trans ローダの有効性の評価を行った．評価の結果，3 つの実行バイナリで，リターンアドレスを書き換える入力を与えた場合に，その書き換えを防ぐことを確認した．しかし，明示的に-fomit-frame-pointer オプションを指定した場合，CVE-2017-14493 は防ぐことができなかった．原因については現在調査中であるが，実装不備が原因と考えられる．

3.2 パフォーマンスの評価

SPEC CPU2006 が提供する 11 個の実行ファイルを用いて，改善前の Safe Trans ローダを適用した場合，改善後の Safe Trans ローダを適用した場合，Safe Trans ローダを適用しなかった場合のそれぞれの実行時間を測定した．評価環境は，CPU は Intel(R) Xeon(R) CPU E5620@2.40GHz，RAM 8GB において，Ubuntu14.04 LTS 32bit を用いた．また，gcc のバージョンは 4.8.4 及び glibc のバージョン 2.18 を使用した．測定結果を表 1 に表す．

表 1: 対策非適用の場合に対してのオーバーヘッド

実行バイナリ	libunwind[%]	frame pointer[%]
400.perlbench	20.418418	0.038233
401.bzip2	0.153281	-0.278099
403.gcc	5.675290	-1.107851
429.mcf	6.875896	7.156963
445.gobmk	0.248169	0.069860
456.hmmer	0.149661	-0.003986
458.sjeng	0.019337	-0.130536
462.libquantum	1.731311	0.796157
464.h264ref	145.724135	0.235106
473.astar	-0.169256	-0.684197
483.xalancbmk	1.440513	0.710484

改善後の Safe Trans ローダを適用した際のオーバーヘッドは，Safe Trans ローダを適用しなかった時に比べ，平均して約 16.57 %増であった．改善前の Safe Trans ローダを適用した際のオーバーヘッドは，平均して約 0.61 %増であった．464.h264ref は，他の実行バイナリに比べて境界検査関数の呼び出し関数が多いので，オーバーヘッドが高くなったと考えられる．

4 まとめ

本論文では，Safe Trans ローダの改善と評価を行った．その結果，Safe Trans ローダはフレームポインタのない実行バイナリに対しても SBoF 攻撃を防ぐことができた．また，適用時のオーバーヘッドは非適用時と比べ，約 16.57 %増であることがわかった．

今後の課題として，実行時のオーバーヘッドの削減および実装不備の修正がある．

参考文献

[1] CWE-121: Stack-based Buffer Overflow, https://cwe.mitre.org/data/definitions/121.html

[2] 菅原 捷汰，王 氷，宮崎 博行，近藤 秀太，渡辺 亮平，横山 雅展，斎藤 孝道:プログラムローダを用いた Stack-based Buffer Overflow 攻撃を緩和する手法の改善と評価，情報処理学会 第 79 回 全国大会

[3] The libunwin project, http://www.nongnu.org/libunwind/

[4] CVE-2013-4256, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-4256

[5] CVE-2017-14493, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-14493