1W-08

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

菅原 捷汰† 王 氷‡ 宮崎 博行‡ 近藤 秀太‡ 渡辺 亮平‡ 横山 雅展† 齋藤 孝道†
明治大学† 明治大学大学院‡

1.はじめに

Stack-based Buffer Overflow（以降，SBoFと呼ぶ）脆弱性（CWE-121）[1]を悪用するSBoF攻撃は，サービスのクラッシュや端末の制御の奪取を許してしまう可能性がある．これまでに，SBoF攻撃に向けて様々な対策技術が考案されてきた．我々は，先行研究[2]において，OSに低依存なアプリケーションとしてのプログラムローダ（以降，Safe Transローダと呼ぶ）を用いて，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に置換することで，SBoF攻撃を緩和する手法を提案した．

本論文では，Safe Transローダの改善，およびより詳細なその評価を行った．具体的な改善としては置換対象とするライブラリ関数の種類を増やした．より詳細な評価の一つとしてはSPEC CPUを用いてパフォーマンスを測定した．その結果，Safe Transローダ適用時のオーバーヘッドはSafe Transローダ非適用時と比べて約0.59%であった．

本論文において，Safe Transローダは，32bit Linux OSにおけるELFの実行バイナリへの適用を想定している．

2.提案方式の概要

2.1. Safe Transローダ

本論文で扱うSafe Transローダは，OS標準のローダに代わってSBoF攻撃の対策を適用したいバイナリ（以降，保護対象バイナリと呼ぶ）をSafe Trans ローダ自身の仮想メモリにロードする．その際に，Safe Transローダは SBoF脆弱性を招くライブラリ関数を，オーバーフローを防ぐために境界検査処理を追加したより安全な関数（以降，境界検査関数と呼ぶ）に置換する．Safe Transローダでは保護対象バイナリのソースコードを必要としない．よって，すでに配布された保護対象バイナリに対策を適用できる．また，Safe Transローダはアプリケーションとして動作するので，組み込み系OSやレガシーな環境など新たなOSに切り替えることが難しい場合でも適用できると期待される．

Safe Transローダの詳細な動作について説明する．Safe Transローダは実行時にコマンドライン引数として保護対象バイナリのパスを受け取る．OS標準のローダによって仮想メモリ上の0x02000000以降にロードされたSafe Transローダは，コマンドライン引数で指定された保護対象バイナリをヒープ領域に読み込む．次に，Safe Transローダは保護対象バイナリのELFヘッダを解析して，保護対象バイナリの各種セグメントの情報やエントリポイントを取得する．ここで取得した情報により，Safe Transローダは保護対象バイナリのLOADセグメントをSafe Trans ローダ自身の仮想メモリにマップする．その後，Safe Transローダは保護対象バイナリのDYNAMICセグメントの情報に基づいて保護対象バイナリの実行に必要な共有ライブラリのロードと再配置処理を行う．再配置処理の際，Safe Transローダは SBoF脆弱性を招くライブラリ関数を，Safe Transローダ上で定義されている境界検査関数に置換する．最後に，保護対象バイナリのエントリポイントに制御を移し，その実行を開始する．

2.2. 境界検査関数

Safe Transローダにより置換された境界検査関数の概要と境界検査の仕組みについて説明する．境界検査関数は，strcpy関数などのSBoF脆弱性を招くライブラリ関数とは違い，書き込み先のバッファの上限サイズ（以降，上限サイズと呼ぶ）を決定し，書き込む文字列のサイズが上限サイズを上回るかどうかを検査する関数である．境界検査関数は，Libsafe[3]の手法を参考に我々が独自で実装した．

境界検査関数が呼び出されると，最初に，上限サイズを決定する．ここで，上限サイズは，書き込み先のバッファの先頭からバッファと同じスタックフレーム内のフレームポインタの先頭の位置までとする（図1）．次に，決定した上限サイズと書き込む文字列のサイズを比較する．書き込む文字列のサイズが上限サイズよりも大きい場合，Safe Transローダはエラーを出力し，保護対象バイナリの実行を終了する．それ以外

Mitigation of Stack-based Buffer Overflow Attack by Program Loader
†Sugawara Shota ‡Wang Bing ‡Miyazaki Hiroyuki ‡Kondo Shuta ‡Watanabe Ryohei
†Yokoyama Masahiro †Saito Takamichi
†Meiji University ‡Graduate School of Meiji University

の場合は書き込み先のバッファに文字列を書き込む.

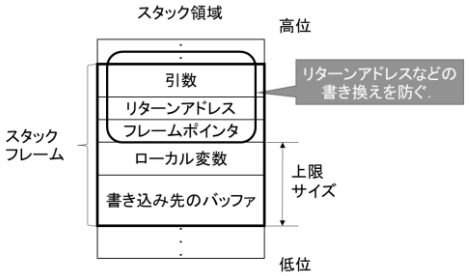

図 1　境界検査の概念図

境界検査関数に，書き込む文字列として上限サイズを上回るサイズの文字列が与えられたとしても，フレームポインタ以降のデータの書き換えを防ぐことが可能である（図 1）.

先行研究[2]では 14 個のライブラリ関数を置換対象の関数に選定したが，本論文では文献[4]を参考に，新たに 7 個のライブラリ関数を置換対象の関数に選定した（表 1）.

表 1　追加した置換対象のライブラリ関数

No	関数のプロトタイプ
1	int fscanf(FILE *stream, const char *format, ...)
2	int sscanf(const char *str, const char *format, ...)
3	int vscanf(const char *format, va_list ap)
4	int vsscanf(const char *str, const char *format, va_list ap)
5	int vfscanf(FILE *stream, const char *format, va_list ap)
6	ssize_t read(int fd, void *buf, size_t count)
7	char *fgets(char *s, int len, FILE *stream)

3.評価

3.1.有効性の評価

CWE-121 のページ[1]で公開されているサンプルコード Example 1 および Example 2 と，CVE-2013-4256[5]で報告されている Network Audio System1.9.3 の 3 種類のバイナリを用いて，Safe Trans ローダの有効性の評価を行った．評価方法は，それぞれのバイナリを Safe Trans ローダ上で実行し，リターンアドレスを書き換えるような入力を与えた場合，リターンアドレスの書き換えを防ぐことができるかを検証するというものである．評価の結果，Safe Trans ローダはリターンアドレスの書き換えを防ぐことができた.

3.2.パフォーマンスの評価

SPEC CPU2006 が提供する 11 個のバイナリを用いて，Safe Trans ローダを適用した場合と適用しなかった場合で，それぞれ実行時間を測定した．評価環境は，CPU は Intel(R) Xeon(R) CPU E5620@2.40GHz，OS は Ubuntu 14.04 LTS 32bit，カーネルのバージョンは 3.13.0-24-generic，gcc/g++のバージョンは 4.8.4 である．測定結果のグラフを図 2 に示す.

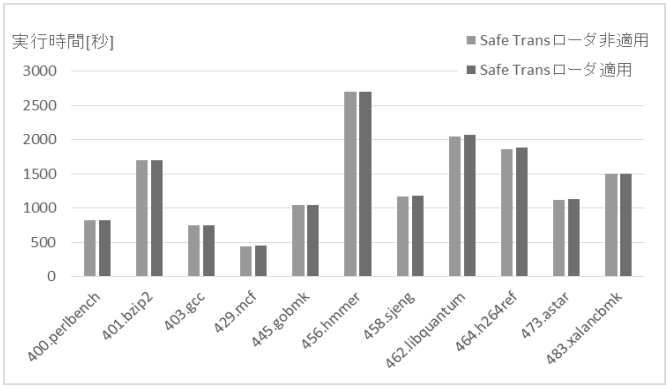

図 2　各バイナリの実行時間

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と比べ，平均して約 0.59%であった.

4.まとめ

本論文では，Safe Trans ローダ[2]の改善と SPEC CPU2006 ベンチマークを用いたより詳細な評価を行った．改善としては置換するライブラリ関数を 7 つ増やした．評価としては改善後の Safe Trans ローダの有効性とパフォーマンスの検証を行った．その結果，Safe Trans ローダは置換対象の関数を起点とした SBoF 攻撃を防ぐことができることと，適用時のオーバーヘッドが非適用時と比べ，約 0.59%であることがわかった.

今後の課題としては，対象バッファとフレームポインタの間にあるローカル変数の改ざん防止がある.

5.参考文献

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

[2] 近藤 秀太，角田 佳史，堀 洋輔，馬場 隆彰，宮崎 博行，王 氷，渡辺 亮平，齋藤 孝道: プログラムローダにより Stack-based Buffer Overflow 攻撃を抑制する手法の提案と実装，情報処理学会 第 78 回 全国大会

[3] Arash Baratloo, Navjot Singh, and Timothy Tsai, “TRANSPARENT RUN-TIME DEFENSE AGAINST STACK SMASHING ATTACKS”, In Proceedings of 2000 USENIX Annual Technical Conference

[4] John Vieg，Gary McGraw，齋藤孝道，河村政雄，武舎広幸，「Building Secure Software - ソフトウェアセキュリティについて開発者が知っているべきこと」，オーム社，2006

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

[6] Robert C. Seacord，歌代和正，久保正樹，椎木孝斉，「C/C++セキュアコーディング 第 2 版」，アスキー・メディアワークス，2014

[7] 齋藤孝道，角田佳史，堀洋輔，馬場隆彰，宮崎博行，王氷，渡辺亮平，近藤秀太: プログラムローダを用いたメモリ破壊攻撃群への対策技術の提案と実装，SCIS2016 暗号と情報セキュリティシンポジウム, 2016