2W-03

プログラムローダにより Stack-based Buffer Overflow 攻撃を緩和する手法の提案と実装

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†

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]への様々な対策技術が，OS，コンパイラ，リンカ，またはライブラリにおいて考案されてきた．しかし，既存の対策技術には，様々な課題がある．

本論文では，アプリケーションプログラムローダを用いたアプローチによる対策を提案する．このアプローチでは，既に配布された実行ファイルに適用できる．より具体的には，SBoF 脆弱性を招くライブラリ関数群を，スタック領域の保護機能を加えたより安全な関数群（以降，境界検査関数群と呼ぶ）に，プログラム実行時に置換する手法の提案，実装，および評価を行う．なお，境界検査関数群の実装は Libsafe[2]を参考にした．

今回想定する環境は，32/64 ビット Linux OS および実行ファイルの形式は ELF である．

2. 既存の対策技術

SBoF 脆弱性の対策技術として，関数を置換する手法を用いる Libsafe，_FORTIFY_SOURCE[3]，および SafeSTR[4]がある．

Libsafe は，ライブラリとして利用し，配布された実行ファイルに適用できる．SBoF 脆弱性を含むライブラリ関数群の呼び出しをプログラムの実行時に検知し，安全な関数に置換する．置換された関数群は，元の関数の機能を有し，さらに，SBoF 脆弱性を検知する．しかし，Libsafe は，既に開発が終了しており，64bit に対応していない．また，ライブラリの依存関係により実行ファイルに適用できない場合がある．

_FORTIFY_SOURCE は，GCC（version4.0 以降）コンパイラでの機能検査マクロのひとつである．コンパイル時に SBoF 脆弱性を含むライブラリ関数群を安全な関数で置換し，実行時に SBoF 脆弱性を検知する．

SafeSTR は，ライブラリとして利用し，ソースコード中で安全なライブラリ関数群を呼び出すことで利用出来る．ヒープにバッファを確保することで，SBoF 脆弱性を発生させない．

_FORTIFY_SOURCE および SafeSTR は，ソースコードを必要とするので，既に配布された実行ファイルに適用できない．

3. 提案方式

3.1 概要

本論文の提案方式は，実行ファイルに適用できるプログラムローダを用いて，保護対象のアプリケーションプログラム（以降，対象プログラムと呼ぶ）に存在する SBoF 脆弱性を含むライブラリ関数群を，境界検査関数群で置換する．この境界検査関数への置換は，この置換を提案の一部として，プログラムローダ（以降，Safe Trans ローダと呼ぶ）[5][6]を用いた．また，SBoF 脆弱性を含むライブラリ関数群は，C/C++文献[7]および Libsafe を参考にし，表 1 に示す 14 個の関数を置換対象とした．

3.2 Safe Trans ローダ

Safe Trans ローダは，引数として受け取った対象プログラムを仮想メモリに展開し，実行する．Safe Trans ローダは，仮想メモリ上のアドレス 0x02000000 から展開され，同仮想メモリ上のヒープ領域に対象プログラムを展開する．そして，対象プログラムの ELF 情報の解析後，対象プログラムの LOAD/DYNAMIC セグメントは仮想メモリに展開される．さらに，脆弱性を含むライブラリ関数群は，Safe Trans ローダに定義されている境界検査関数群に置換される．そして，対象プログラムの実行が開始される．

3.3 境界検査関数群の仕組み

Safe Trans ローダは，SBoF 脆弱性を含むライブラリ関数群のバッファへの書き込み可能範囲を，書き込み先アドレスから格納されたフレームポインタ（以降，格納フレームポインタと呼ぶ）の先頭アドレスまでとする．これにより，格納フレームポインタおよび格納されたリターンアドレスの改竄を起点とする攻撃群を防ぐことができる．

続いて，図 1 を元に具体的な境界検査関数群の仕組みを解説する．図 1（a）は，SBoF 脆弱

Proposal and Implementation of Program Loader Mitigating a Stack-based Buffer Overflow Attack
†Kondo Shuta ‡Sumida Yoshifumi ‡Hori Yosuke
‡Baba Takaaki ‡Miyazaki Hiroyuki ‡Wang Bing
†Watanabe Ryohei †Saito Takamichi
†Meiji University ‡Graduate School of Meiji University

性を含むライブラリ関数群を使用し，バッファーオーバーフローが起こった様子である．スタック領域に確保されたバッファを超えた書き込みが可能なので，格納フレームポインタおよび格納されたリターンアドレスの書き換えが可能なことがわかる．他方，提案方式の境界検査関数を使用した際，格納フレームポインタより高位への書き込みを防ぐ（図１（ｂ））．

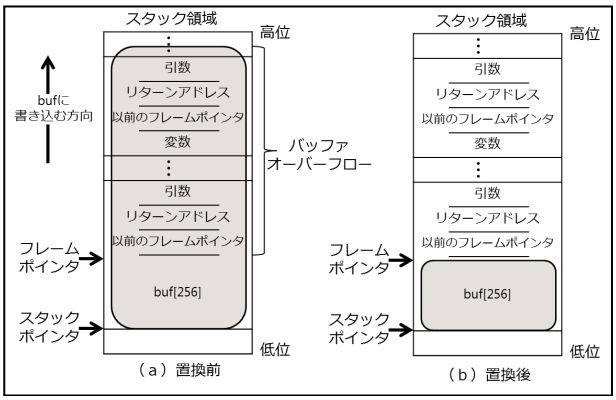

図１ 提案方式を適用前後のスタック領域

動作の例として，境界検査関数の一つであるHstrcpy 関数呼び出し時の手順を示す．

イ） Hstrcpy 関数呼び出し時のフレームポインタの値を参照し，書き込み先アドレスがあるスタックフレーム内の格納フレームポインタの先頭アドレスを求める．

ロ） 書き込み先アドレスがあるスタックフレーム内の格納フレームポインタの先頭アドレスと，書き込み先アドレスの差を書き込み可能範囲とする．

ハ） 書き込み可能範囲が書き込み元サイズ以上であれば，文字列をコピーする．そうでなければ，対象プログラムは終了する．

4. 評価

4.1 Safe Trans ローダの有効性

CWE-121 を記載する Mitre 社サイトに例示された SBoF 脆弱性を含むプログラム Example 1 および Example 2 を用いて提案方式の有効性を示す．結果として，Safe Trans ローダ上の境界検査関数群によって，Example1 および Example 2 実行時に，格納フレームポインタより高位への書き込みを検知できた．

4.2 オーバーヘッド

本論文では，Linux の標準の ELF ローダおよび Safe Trans ローダ環境において，ライブラリ関数群に入力値として 2,048bytes のサイズの文字列を渡すプログラムを 1,000 回実行した．その際に，ライブラリ関数の実行処理時間を gettimeofday 関数を用いて計測し，平均を求めた結果を表 1 に示す．

評価環境は，Intel(R) Xeon(R) CPU E5620@2.40GHz，および Ubuntu14.04 LTS 32bit とした．

結果として，オリジナルのライブラリ関数に対応する境界検査関数の実行処理時間は約 3μs 以内の増減なので，プログラムに与える影響は小さいと言える．

表1 各ローダでのライブラリ関数群の実行処理時間

Linuxの標準ローダ		Safe Transローダ	
関数名	実行処理時間	関数名	実行処理時間
memcpy	3.623μs(0%)	Hmemcpy	3.706μs (2%)
strcpy	3.339μs (0%)	Hstrcpy	4.427μs (33%)
strncpy	363.789μs (0%)	Hstrncpy	360.493μs (−1%)
strcat	6.081μs (0%)	Hstrcat	4.384μs (−28%)
strncat	5.137μs (0%)	Hstrncat	5.473μs (7%)
stpcpy	3.437μs (0%)	Hstpcpy	4.335μs (26%)
gets	17.932μs (0%)	Hgets	18.392μs (3%)
getwd	4.296μs (0%)	Hgetwd	3.143μs (−27%)
realpath	4.593μs (0%)	Hrealpath	5.449μs (19%)
sprintf	14.315μs (0%)	Hsprintf	12.969μs (−9%)
snprintf	14.602μs (0%)	Hsnprintf	12.751μs (−13%)
vsprintf	13.349μs (0%)	Hvsprintf	12.715μs (−5%)
vsnprintf	13.325μs (0%)	Hvsnprintf	13.045μs (−2%)
scanf	32.603μs (0%)	Hscanf	30.93μs (−5%)

5. まとめ

本論文では，アプリケーションプログラムローダを用いて，SBoF 攻撃を緩和する手法，実装，および評価を行った．効果及びパフォーマンスもよいことが確認できた．また，このアプローチでは，既に配布された実行ファイルに適用できることがメリットである．

今後の課題としては，メモリ破損脆弱性群の報告がされている様々な実行ファイルに対しての検証およびベンチマークツールを用いた詳細なパフォーマンスの評価がある．

6. 参考文献

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

[2] Baratloo, Arash, Navjot Singh, and Timothy K. Tsai. "Transparent Run-Time Defense Against Stack-Smashing Attacks." *USENIX Annual Technical Conference, General Track*. 2000.

[3] FEATURE_TEST_MACROS(7) http://man7.org/linux/man-pages/man7/feature_test_macros.7.html

[4] バッファオーバーフロー: #2 ソースコード記述時の対策 https://www.ipa.go.jp/security/awareness/vendor/programmingv2/contents/c902.html

[5] 齋藤 孝道† 堀 洋輔‡ 角田 佳史‡ 馬場 隆彰‡ 宮崎 博行‡王 永‡ 近藤 秀太† 渡辺 亮平†: プログラムローダにより UAF 攻撃を抑制する手法の提案と実装

[6] 齋藤孝道，上原崇史，金子洋平，鈴木舞音，角田佳史，堀洋輔，馬場隆彰，宮崎博行: リリースされたバイナリに適用するスタックベース BoF 攻撃緩和技術の試作と評価 Symposium on Cryptography and Information Security 2015

[7] Building Secure Software John Viega, Gary McGraw 共著 齋藤孝道 監訳/武舎広幸・河村政雄 共役

[8] 齋藤孝道：マスタリング TCP/IP 情報セキュリティ編，オーム社，2013.