5X-04

バッファオーバーフロー対策技術の実現アプローチに基づく調査と分類

宮崎 博行† 金子 洋平‡ 鈴木 舞音‡ 上原 崇史‡ 角田 佳史‡ 堀 洋輔‡ 馬場 隆彰† 齋藤 孝道†

明治大学†　　明治大学大学院‡

1.概要

ソフトウェアの脆弱性の一つに古くから存在する Buffer Overflow[1]がある．この脆弱性は 1972 年に Computer Security Technology Planning Study[2]で文書として初めて公開された．脆弱性データベースの 1 つである NVD（National Vulnerability Database）には，現在に至るまでに Buffer Overflow は二番目に多く報告されており，現在も報告が絶えない．

現在では，OS，コンパイラもしくはリンカに様々な防御・攻撃緩和策（以降，対策技術という）が，対策技術としてある．しかし，Linux のディストリビューションやコンパイラのバージョンによっては，利用者が自ら適用しなければならない対策技術がある．利用者が対策技術を把握し，自らそれを適用することは，脆弱性の緩和に繋がると考えられる．

そこで，本論文では現在までに報告されている Buffer Overflow を悪用する攻撃への対策技術の実現アプローチに基づく調査と分類を行う．また，利用者が自ら適用しなければならない対策技術に関しては適用方法を示す．

2.Buffer Overflow とは

Buffer Overflow は，C/C++などで作成されたプログラムに対して，メモリ上に確保したバッファを超えるデータの入力を受け付けてしまう脆弱性もしくは現象である．この脆弱性を悪用して，攻撃者はスタック領域やヒープ領域に不正な命令コードを書き込み，実行する．これにより，不正な権限の取得やシステムへの侵入を許してしまう[1].

Buffer Overflow を悪用する攻撃において，スタック領域を対象とした攻撃は Stack-based Buffer Overflow 攻撃，ヒープ領域を対象とした攻撃は Heap-based Buffer Overflow 攻撃と呼ばれる．前者を応用した攻撃には ROP（Return-Oriented Programming）攻撃[3]や Return-to-libc 攻撃[3]などが挙げられる．後者を応用した攻撃には Double Free 攻撃[3]や Heap Spray 攻撃[3]が挙げられる．

3.既存の対策技術

3.1. OS における対策技術

3.1.1. 主な対策技術

OS の主な対策技術は，データ領域におけるコード実行防止機能[4]，ASCII-armor[4]，ASLR（Address Space Layout Randomization）[4]及び kASLR（Kernel Address Space Layout Randomization ）[5]が挙げられる．これらの対策技術は，OS の設定を変更することにより，プログラムの実行時に機能を適用することができる．

データ領域におけるコード実行防止機能は，スタック，ヒープ，.data 及び.bss といった領域におけるコード実行を禁止する機能である．

ASCII-armor は，32bitOS の場合，共有ライブラリを 0x00ffffff 以下のアドレスへ再配置する機能である．これは，Return-to-libc などの共有ライブラリのアドレスを指定する攻撃を困難にする．この機能は CentOS ではデフォルトで適用されている．それ以外の環境では prelink --exec-shield により再リンクする必要がある．

Linux における ASLR は，プログラムのメモリ配置の基底アドレスを実行時にランダム化する機能である．加えて，PIE 形式でコンパイルされたコードでは，テキスト領域の基底アドレスもランダム化できる．この機能は，ROP 攻撃などのアドレスを指定する攻撃を困難にする．

kASLR は，カーネル起動時に，カーネルイメージの展開先とモジュール読み込み位置をランダム化する機能である．カーネルイメージを展開する領域は 512M から 1G に拡張され，その範囲内にカーネルイメージが展開される．

3.1.2. Heap Protector

Heap Protector[6]とは，free 関数の呼び出し時に，Buffer Overflow の検知や free 関数による二重解放の有無の確認を行う glibc の機能で

A Survey of Approach to Mechanism of Prevention/Mitigation against Buffer Overflow Attacks

†Hiroyuki MIYAZAKI ‡Yohei KANEKO
‡Maine SUZUKI ‡Takafumi UEHARA
‡Yoshifumi SUMIDA ‡Yosuke HORI †Takaaki BABA
†Takamichi SAITO
†Meiji University ‡Graduate School of Meiji University

ある．環境変数 MALLOC_CHECK_に 1 から 3 の値を設定した場合に機能する．

MALLOC_CHECK_に 1 以上の値が定義されている場合，エラー検出用のマジックバイトを確保したヒープ領域の直後に挿入する．また，確保したヒープ領域の解放時に，mem2chunk_check 関数で，次の二つの値の改竄チェックを行う．一つ目は malloc で割り当てられたメモリの直前にある管理情報，二つ目はマジックバイトである．これらの値の改ざんが行われているかつ MALLOC_CHECK_の値が 1 の場合はエラーメッセージの表示，2 の場合は プログラムの停止，3 の場合はエラーメッセージを表示し，プログラムを停止する．

3.2. コンパイラにおける対策技術

3.2.1. 主な対策技術

コンパイラにおける主な対策技術は，SSP[4]（Stack Smashing Protector）や Automatic Fortification[4]が挙げられる．これらの対策技術は，プログラムのコンパイル時にオプションを指定することで，適用する．

SSP は，フレームポインタの低位にカナリアと呼ばれるランダム値を挿入し，カナリアの改ざんを検知する機能である．カナリアの改竄が検知された場合は，プログラムを停止する．

Automatic Fortification は，Buffer Overflow 攻撃に悪用される関数のチェック機能を追加した関数に置換し，Buffer Overflow を検知する機能である．この機能は，コンパイルオプションに D_FORTIFY_SOURCE=1 のように 1 以上の値を指定することで機能する．

3.2.2. Memory Sanitizer

Memory Sanitizer[7]は，C/C++で作成された PIE 形式のプログラムにおいて，スタック領域とヒープ領域内の未初期化の変数や配列内の値の参照を検知する機能である．

この機能は Shadow Mapping と呼ばれる機構を用いて Application 領域と Shadow 領域を確保する．Application 領域には保護対象のデータ，Shadow 領域には実行中の命令を格納するレジスタの値が格納される．Application 領域の値と Shadow 領域の値による演算結果が「0」の場合には初期化済みを表し，「1」の場合は未初期化を表す．未初期化の場合には，プログラムを停止する．

Memory Sanitizer を有効にすると，実行時間が無効時の 2 倍から 3 倍に増加する．この機能は，コンパイルオプションに-fsanitize=memory を指定することで機能する．

3.2.3. Address Sanitizer

Address Sanitizer[8]は，C/C++で作成されるプログラムにおいて，確保済みメモリ領域の要素外参照や free 関数による二重解放の検知を行う機能である．この機能は以下の処理を行う．

(1)メモリアクセス処理を，Shadow Mapping に置換する．未初期化なメモリの参照が検知されると，プログラムを停止する．

(2)隣接するメモリ間に redzone と呼ばれる領域を確保し，malloc 関数と free 関数を，redzone の改竄を検知する関数に置換する． redzone の改竄を検知すると，プログラムを停止する．

この機能は，コンパイルオプションに-fsanitize=address を指定することで機能する．

3.3. リンカにおける対策技術

リンカにおける対策技術は，RELRO[4]（RELocation Read-Only）が挙げられる．この対策技術は，プログラムのコンパイル時にリンカオプションを渡すことで，適用する．

RELRO は，GOT 領域と呼ばれる動的リンクされる関数のアドレスを解決するための領域などを，読み取り専用にする機能である．加えて，遅延バインドと呼ばれる機能を無効にし，RELRO を有効にすることで，GOT 領域を書き換える攻撃を防ぐことができる．

4. まとめ

本論文では，Buffer Overflow 攻撃への対策技術が OS，コンパイラ及びリンカのどの段階で適用できるかを纏め，分類を行った．OS における対策技術は，OS の設定を変更することで適用できる．コンパイラやリンカにおける対策技術は，コンパイルやリンク時にオプションを指定することで適用できる．また，本論文で述べた対策技術の Address Sanitizer，Memory Sanitizer，Heap Protector 及び ASCII-armor は自ら適用する必要がある対策技術である．これらの対策技術の機能を把握し，適用することで脆弱性を緩和することができる．

5. 参考文献

[1] 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
[2] NIST，”Computer Security Technology Planning Study” http://csrc.nist.gov/publications/history/ande72.pdf
[3] 鈴木舞音，上原崇史，金子洋平，堀洋輔，馬場隆彰，斎藤孝道，メモリ破損脆弱性に対する攻撃の調査と分類，コンピュータセキュリティシンポジウム 2014 論文集，pp.767-774
[4] 齋藤孝道，鈴木舞音，上原崇史，金子洋平，角田佳史，馬場隆彰，メモリ破損攻撃への対策技術の調査と分類，コンピュータセキュリティシンポジウム 2014 論文集，pp.775-782
[5] Kernel Address Space Layout Randomization http://selinuxproject.org/~jmorris/lss2013_slides/cook_kaslr.pdf
[6] malloc(3) のメモリ管理構造　VA Linux Systems Japan 株式会社 http://www.valinux.co.jp/technologylibrary/document/linux/malloc0001
[7] Evgeniy Stepanov, Kostya Serebryany,”MemorySanitizer”Apr 29, 2013 http://llvm.org/devmtg/2013-04/stepanov-slides.pdf
[8] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, Dmitry Vyukov, ”AddressSanitizer: A Fast Address Sanity Checker”