L-028

バッファオーバーフロー攻撃に関する防御技術の調査
A Survey of Mechanism of Protection/Mitigation against Buffer Overflow Attacks

角田 佳史† 金子 洋平† 鈴木 舞音† 上原 崇史† 齋藤 孝道‡
Yoshifumi SUMIDA Yohei KANEKO Maine SUZUKI Takafumi UEHARA Takamichi SAITO

1. 概要

1.1 はじめに

バッファオーバーフロー[1]脆弱性は，現在でも CVE (Common Vulnerabilities and Exposures) や NVD (National Vulenrability Database) といった脆弱性データベースに報告されている．

NVD が 1990 年から現在までに報告しているバッファオーバーフローの件数は 6,808 件である．また，CVE でのバッファオーバーフローの報告件数は 1,040 件の報告がある．NVD においては，バッファオーバーフローの報告件数は攻撃全体の 1 割を占めており，報告件数の推移をみると減ってきているが，現在も重大なセキュリティの課題である．

そこで，本論文では，現在までに報告されているバッファオーバーフロー脆弱性を悪用する攻撃(以下，BoF 攻撃という) に対する数々の保護機能の調査と分類をする．

1.2 歴史

バッファオーバーフローは，1972 年に Computer Security Technology Planning Study[1]にて初めて公に文書化された．また，バッファオーバーフローを悪用した攻撃に関する文書は，1988 年に公開され，Morris Worm[10]を始めとする数々のバッファオーバーフローを悪用した攻撃が報告されるようになった．その後，BoF 攻撃に対しての保護機能が登場したが，それらを回避する攻撃の報告も行われてきており，いまだに根本的な解決に至っていない．

2. バッファオーバーフローとは

本来，バッファオーバーフローとは，C/C++などのプログラムにおいて用意したメモリ内のバッファ領域に対して，そのサイズを超えるデータを受け付けてしまう脆弱性のことである．この脆弱性を悪用することで，不正なコードの実行，不正な権限の取得，及びシステムへの侵入の足がかりを攻撃者に与えてしまう．この脆弱性を用いた攻撃として，スタックオーバーフロー攻撃[5]，ヒープオーバーフロー攻撃[5]，Off-by-one 攻撃[6]，GOT 書き換え攻撃[7]，Return-to-libc 攻撃 [2] ， ROP (Return-Oriented Programming) 攻撃[3] (以降，ROP 攻撃と呼ぶ) ，JOP (Jump-Oriented Programming) 攻撃[4]，String-Oriented Programming 攻撃 [8] ， Blind Return-Oriented Programming 攻撃[9]が挙げられる．これら攻撃の概要については， [12]にて報告する．

† 明治大学大学院 Graduate of Meiji University
‡ 明治大学 Meiji University

3. 既存の保護機能

3.1 OS における保護機能

3.1.1 データ領域におけるコード実行防止機能

プログラムが使用するメモリ内におけるスタック，ヒープ，data 及び bss の各領域上でのコードの実行を不可能にする機能である．この機能は，Windows では DEP (Data Execution Prevention) ，Linux では Exec shield もしくは PaX と呼ばれる．

DEP は，Windows XP Service Pack2 で導入された．Windows の DEP には，プロセッサの機能である NX (No eXecute) /XD (eXecute Disable) ビットを利用するハードウェア DEP と，利用しないソフトウェア DEP (SafeSEH) があるが，コード実行防止機能は前者にあたる．

Exec shield は，2002 年後半に Red Hat 社が開始したプロジェクトである．プロジェクト開始当初は，Linux カーネル 2.4.21-rc1 に対するセキュリティパッチとして公開されていたが，Linux カーネル 2.6.8 以降でメインストリームにマージされた．また，Exec shield には，Windows と同様に NX/XD ビットを利用したデータ領域でのコード実行防止機能に加え，ASCII Armor と呼ばれる機能も含まれており，Ubuntu においては Exec Shield とは独立した機能として有効になっている．これは，共有ライブラリをメモリ上に配置する際に，(32 bit 版の OS では) 0x00ffffff 以下のアドレスへ配置する機能である．strcpy 等の共有ライブラリに含まれる文字列操作関数は NULL (0x00) 文字を文字列の終端と見なすため，いくつかの攻撃を防ぐことができる．

PaX は，2000 年に初めてパッチがリリースされ，現在では従来の開発者達により保守が続けられている．加えて，この機能は，3.1.2 節で解説するアドレスのランダマイズ機能も含んでいる．

表 1 で，現在の Linux ディストリビューションの Exec shield，PaX 及び ASCII Armor の対応状況を示す．

表 1 Exec shield，PaX 及び ASCII Armor の対応状況

ディストリビューション	対応状況	保護機能
CentOS	3 以降	Exec shield ASCII Armor
Debian	Option	Exec shield ASCII Armor
Fedora	Core1 以降	Exec shield, ASCII Armor /PaX
Gentoo	Option	PaX
Red Hat Enterprise Linux	3 以降	Exec shield ASCII Armor

3.1.2 アドレスのランダマイズ機能

プログラムに割り当てられたメモリ空間において，スタック，ヒープ，data 及び bss として利用する領域の基底

アドレスをランダム化する機能があり，ASLR（Address Space Layout Randomization）と呼ばれる．これにより，Return-to-libc 攻撃などを防ぐことができる．ただし，PIE（Position -Independent Executable)でコンパイルする事により有効となる．Windows における ASLR は，Vista 以降に導入されており，標準で有効になっているが，コンパイラで /DYNAMICBASE と明示的に有効にすることもできる．また，Linux においては，Linux カーネル 2.6.12 から導入された． Linux では，プログラムの初期実行時のみ，スタック及びヒープとして利用する領域の基底アドレスをランダム化するが，Windows では，8 以降から，プログラムの実行中においても，前述の基底アドレスをランダム化することができるようになった．また，Windows 8 以降においては， Bottom-up/Top-down Randomization やヒープ領域への ASLR が適用されており，特に 64 bit 版では，より広い範囲でランダム化が行われている[13]．一方で，Linux においては，プログラムを PIE でコンパイルすることにより，コード領域のランダム化も可能である．

我々の実験では，64bit 版の Windows 8.1 におけるヒープ上のバッファ領域のアドレス配置のエントロピーは，6.365902552 bit であり，64bit 版の CentOS 6.5 (Kernel 2.6.32-431.7) においては，12.3200363 であった．

3.2 コンパイラ/リンカにおける保護機能

3.2.1 GCC のコンパイルオプション

3.2.1.1 SSP (Stack Smashing Protection)

GCC (GNU Compiler Collection) 導入されているスタックオーバーフロー攻撃から引数やローカル変数及びポインタを保護する機能である．この機能は，IBM により，StackGuard と呼ばれる保護機能を元に開発が行われた．GCC4.1 以前では，パッチにて公開されていたが，4.1 以降では標準で導入されている．これは，フレームポインタのより低位にカナリアと呼ばれるランダムな値を挿入し，その値が関数終了時に改竄されているかどうかをチェックするルーチンをコンパイル時にオリジナルコードに追加する．実行時に改竄を検知した場合には，プログラムを停止する．また，ローカル変数及びポインタをバッファ領域のより低位に再配置する機能や，Off-by-one バグを防ぐために，バッファ領域とカナリアの間に何バイトかのスペースを確保する機能も持つ． ただし，バッファオーバーフローの発生自体を防ぐことはできないといったことや，プログラムの再コンパイルが必要といったデメリットがある．

3.2.1.2 Automatic Fortification

GCC4.0 から標準で導入されたセキュリティ機能で，BoF 攻撃からプログラムを保護することができる．これは，共有ライブラリに含まれるバッファオーバーフローの危険性を持つ関数群を，安全な関数に置換する機能である．この機能は，コンパイル及び実行時に，バッファ領域を超えるサイズの文字列が渡されていないかのチェックを行う．この機能により，リターンアドレスやフレームポインタの改竄を未然に防ぐことができる．また，D_FORTIFY_SOURCE=2 というオプションをコンパイル時に加えることにより，実行中に書式文字列攻撃を防ぐことができる機能も備えている． ただし，ある関数で宣言されているバッファを引数として別の関数に渡し，その別の関数内で引数として渡されたバッファに対して，何かしらの文字列をコピーする際には，この機能は働かない．

3.2.1.3 RELRO (RELocation Read-Only)

仮想メモリ上における GOT（Global Offset Table）領域を読み取り専用にする機能である．ただし，この機能は，共有ライブラリに含まれる関数のアドレス解決を行う.got セクションは読み取り専用になるが，.got.plt セクションは読み込み専用にならない．それゆえに，攻撃者によって.got.plt セクションを利用した GOT 書き換え攻撃が行われる可能性がある．これを防ぐためには，RELRO を使用する際に，標準で有効になっている遅延バインドと呼ばれる機能を無効にする必要がある．この機能を無効にすることにより，プログラムが使用する共有ライブラリに含まれる関数のアドレス解決を実行時に行うことができる．これは，.got.plt セクションを生成しないため，GOT 書き換え攻撃を防ぐことができる．

3.2.2 Visual Studio のコンパイルオプション

3.2.2.1 GS Protection (/GS)

Windows の Visual Studio のコンパイラに導入されており，スタックオーバーフロー攻撃からプログラムを保護する機能である．これは，フレームポインタと例外ハンドラフレームの間に GS Cookie と呼ばれるランダムな値(SSP のカナリアと同じようなもの）を挿入し，関数終了時にその値が改竄されているかどうかをチェックする．改竄されていた場合は，プログラムを停止する．また，引数をバッファ領域のより低位にコピーする機能も持つ．実際に引数を利用する際には，コピーされたものを利用する．GS Protection は，Visual Studio 2003 に初めて導入され，2005 年版からは，引数の改竄を防ぐ機能が追加された．

4. まとめ

本論文では，いくつかの BoF 攻撃に対する OS 及びコンパイラ/リンカの保護機能について纏め，分類をした．現在でも様々な保護機能が存在するが，それらを回避する新たな攻撃手法も現れており，保護機能の改善，新たな保護機能の考案，及び効果的な組み合わせを考えていく必要があると考えられる．

参考文献

[1] NIST ”Computer Security Technology Planning Study” http://csrc.nist.gov/publications/history/ande72.pdf
[2] Infosec Writers “Bypassing non-executable-stack during exploitation using return-to-libc” http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf
[3] Erik Buchanan(2008) . When Good Instructions Go Bad:Generalizing Return-Oriented Programming to RISC
[4] Tyler Bletsch, Xuxian Jiang, Vince W.Freeh (2011) . Jump-Oriented Programming: A New Class of Code-Reuse Attack.
[5] 愛甲健二 (2006) , ハッカープログラミング大全 攻撃編,
[6] EXPLOIT DATABASE “Off-by-One exploitation tutorial” http://www.exploit-db.com/wp-content/themes/exploit/docs/28478.pdf
[7] ももいろテクノロジー 「format string attack による GOT overwrite をやってみる」 http://inaz2.hatenablog.com/entry/2014/04/20/041453
[8] Mathias Payer, Thomas R. Gross (2013) . String Oriented Programming: When ASLR is not Enoguh.
[9] SECURE COMPUTER SYSTEMS “Blind Return Oriented Programming (BROP) ” http://www.scs.stanford.edu/brop/
[10] Morris Worm, http://ja.wikipedia.org/wiki/Morris_worm http://itpro.nikkeibp.co.jp/members/ITPro/SEC_CHECK/20010907/1
[11] 齋藤孝道 (2013) , マスタリング TCP/IP 情報セキュリティ編, オーム社
[12] 堀 洋輔, 金子 洋平, 鈴木 舞音, 上原 崇史, 齋藤 孝道(2014) . バッファオーバーフロー攻撃の調査と分類
[13] FFRI 「Windows 8 – 脆弱性緩和機能の強化」 http://www.ffri.jp/assets/files/monthly_research/MR201209_Windows8_Exploit_Mitigation.pdf