L-029

バッファオーバーフロー攻撃の調査と分類
A Survey of Type of Buffer Overflow Attacks

堀 洋輔† 金子 洋平† 鈴木 舞音† 上原 崇史† 齋藤 孝道‡
Yosuke HORI Yohei KANEKO Maine SUZUKI Takafumi UEHARA Takamichi SAITO

１． はじめに

ソフトウエアにおいて，バッファオーバーフロー[1]の問題は，古くから存在しており，現在でも NVD（National Vulnerability Database）[2]などの脆弱性データベースに報告が絶えない深刻な脆弱性の一つである．

バッファオーバーフローに関する脆弱性（以降，BoF 脆弱性という）は 1972 年に Computer Security Technology Planning Study[3]にて初めて公に文書化された．その後，BoF 脆弱性は，1988 年に発生した Morris Worm[4]をはじめ，2001 年に発生した Code Red[5]や 2014 年に発生した Adobe Flash Player のマルウェア感染による情報流出[6]など，現在に至るまで，様々な攻撃に悪用されてきている．過去の攻撃手法に対して，OS やコンパイラに防御機能が施されてきているが，防御機能を回避するような攻撃も報告されている．

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攻撃の種々の手法の調査と分類する．

２．バッファオーバーフロー

バッファオーバーフローは，プログラム内に用意したバッファのサイズよりも大きい入力を受け付けてしまう現象である．メモリは低位から高位へ書き込まれるため，溢れたバッファはバッファより高位の部分を上書きしてしまう．このような現象が起こり得る原因は，C/C++などのプログラムにおいて，入力値のサイズの確認忘れであり，スタックやヒープなどのデータ領域で発生する．前者での攻撃を，スタックオーバーフロー攻撃，後者での攻撃を，ヒープオーバーフロー攻撃という．また，その他，後述するデータ領域への攻撃もある．

３．スタックが攻撃対象となる攻撃手法

３．１． スタックオーバーフロー攻撃

関数が呼び出されると，スタック領域に，その関数への引数，リターンアドレス，呼び出した関数のスタックの基底アドレスと呼び出された関数が使用するバッファが高位から低位に向かって順に積まれる．

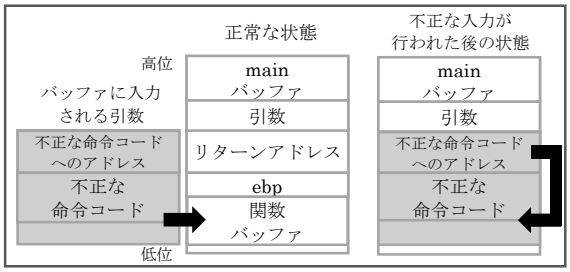

図１．スタックオーバーフロー攻撃の概念

スタックオーバーフロー攻撃[3]は，バッファを溢れさせることで，高位にあるリターンアドレスを，（攻撃者が）埋め込んだ不正な命令コードの開始アドレスに上書きする．これにより，関数のリターン時に，不正な命令コードに実行を移すことができる（図１参照）．

３．２． Return-to-libc 攻撃

Return-to-libc 攻撃[7]は，リターンアドレスを libc といったライブラリ関数へのアドレスに書き換え，関数のリターン時に，攻撃者が所望する関数を呼び出す．

Return-to-libc 攻撃は，スタック内で命令コードの実行を行わないため，データ実行防止機能を回避できる．

３．３． GOT 書き換え攻撃

GOT（Global Offset Table）領域 及び PLT（Procedure Linkage Table）領域は，ELF（Executable and Linkable Format）形式のプログラム実行時の動的リンクの際に共有ライブラリ関数へ間接的に参照するためのアドレスが格納されているデータ領域のことであり，.got 及び.got.plt セクションのことである．

GOT 書き換え攻撃[8]は，GOT 領域及び PLT 領域内の共有ライブラリへの参照アドレスを不正な命令コードへのアドレスに書き換える．これにより，共有ライブラリ関数の実行時に，不正な命令コードが呼び出される．攻撃実現のための一つの方法は，バッファへの書き込みをする命令に使われるポインタのアドレスを，バッファを溢れさせることで，攻撃者が書き換えたい共有ライブラリへの参照アドレスがある GOT 領域及び PLT 領域内へのアドレスに上書きする．これにより，ポインタ書き換え以降のバッファへの書き込みは GOT 領域及び PLT 領域内で行われるため，共有ライブラリへの参照アドレスを攻撃者が用意した不正な命令コードへのアドレスに書き換えることができる（図 2 参照）．

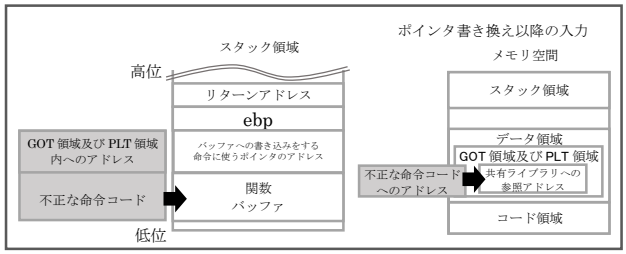

図 2. ポインタを利用した GOT 書き換え攻撃の概念

３．５． Return-to-Register 攻撃

Return-to-Register 攻撃[9]は，上書きするリターンアドレスを，「レジスタが指すアドレスにジャンプする命令が存在するアドレス」に上書きすることで埋め込んだ不正なコードを実行させる．一つの方法は，スタックの先頭アドレスを指す esp レジスタを悪用する．溢れさせたバッファによって，リターンアドレスをコード領域内の esp レジス

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

タが指すアドレスにジャンプする命令が存在するアドレスに書き換え，リターンアドレスの高位側に不正なコードを隣接して埋め込む．これにより，リターン命令が実行されると，esp レジスタが指すアドレスに制御が移り，その後の実行は esp レジスタが指すアドレスからになる．この時，esp レジスタは不正な命令コードの先頭を指しているので，埋め込んだ不正なコードが実行される．Return-to-Register 攻撃の場合，書き換えたアドレスによる不正なコードへの直接のアドレス指定がないため，バッファ領域の確保される範囲がランダムになっても攻撃を行うことができる．

3．5．Return-Oriented Programming 攻撃

Return-Oriented Programming 攻撃[10]は，コード領域内に存在する攻撃者が意図する実行に必要な命令とリターン命令の命令群であるガジェットを，攻撃者が意図する順に実行させる．ガジェットへ実行を移すために，スタック領域内のバッファを溢れさせることで，リターンアドレスを一つ目のガジェットへのアドレスに上書きし，二つ目以降のガジェットをリターンアドレスより高位側に隣接して埋め込む．これにより，関数の終了時にリターン命令が実行されると，一つ目のガジェットが実行される．ガジェットの最後はリターン命令であるので，一つ目のガジェットのリターン命令が実行されると二つ目以降のガジェットも順次実行され，攻撃者が意図する実行が可能となる（図 3 参照）．

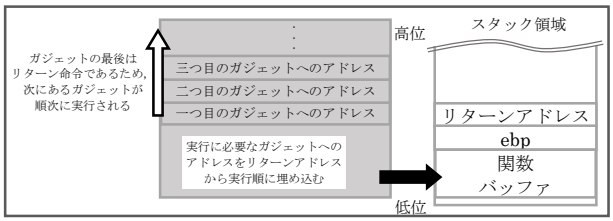

図 3．Return-Oriented Programming 攻撃の概念

4．ヒープが攻撃対象となる攻撃手法

4．1．ヒープオーバーフロー攻撃

ヒープ領域は，バッファと前後のバッファの先頭アドレスでバッファ同士を紐づけるメタデータによる双方向リストとなっている．メタデータ内には，ジャンプアドレスを含む場合がある．

ヒープオーバーフロー攻撃[11]は，関数へのポインタを書き変える手法や，メタデータにあるジャンプアドレスを不正なコード命令へのアドレスに書き換える（図 4 参照）．

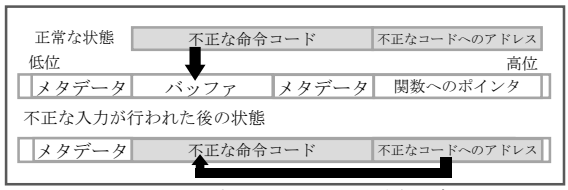

図 4．ヒープオーバーフロー攻撃の概要

4．2．Heap-spraying 攻撃

ヒープオーバーフロー攻撃では，不正なコードが格納されているアドレスを特定する必要がある．

Heap-spraying 攻撃[12]では，不正なコードの前に何も行わずに次の命令に移る NOP 命令を埋め込む．それにより，書き換えた飛び先アドレスが攻撃者の用意した NOP 命令の何れかになれば，不正なコードまで到達する可能性が高まる．

5．防御機能の対応

表 1 は，Off-by-one攻撃[13]，Jump-Oriented Programming 攻撃[14]，String-Oriented Programming 攻撃[15]や Blind Return-Oriented Programming攻撃[16]含めた攻撃手法に対して，OS（CentOS6.4）やコンパイラ（gcc-4.7.2）の防御状況の対応関係をまとめたものである．丸表記されている攻撃は，防御機能により防ぐことが出来る攻撃である．

6．まとめ

本論文では，いくつかのバッファオーバーフローを悪用した攻撃手法についてまとめ，分類した．これら以外にも新たな攻撃手法も出ているので，今後はバッファオーバーフロー攻撃の対象や不正なコードを実行する方法ごとによって，より細かい分類が必要となると考えられる．

7．参考文献

[1] 齋藤孝道著(2013),マスタリング TCP/IP 情報セキュリティ編,オーム社.
[2] National Vulnerability Database,http://nvd.nist.gov/
[3] NIST ”Computer Security Technology Planning Study”, http://csrc.nist.gov/publications/history/ande72.pdf
[4] Morris Worm, http://ja.wikipedia.org/wiki/Morris_worm
[5] Code Red, http://ja.wikipedia.org/wiki/Code_Red
[6] CVE-2014-0515, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0515
[7] Infosec Writers,“Bypassing non-executable-stack during exploitation using return-to-libc” http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf
[8] How to hijack the Global Offset Table with pointers for root Shells, http://www.infosecwriters.com/text_resources/pdf/GOT_Hijack.pdf
[9] backyard 「BoF+ASLR バイパス(ret2reg)」, http://backyard.blog.jp/archives/2057528.html
[10] Erik Buchanan(2008), When Good Instructions Go Bad:Generalizing Return-Oriented Programming to RISC
[11] 愛甲健二著(2006),ハッカープログラミング大全 攻撃編,データハウス.
[12] マイナビニュース「PoC の内側 - Heap Spray」, http://news.mynavi.jp/column/itsecurity/011/
[13] EXPLOIT DATABASE “Off-by-One exploitation tutorial” http://www.exploit-db.com/wp-content/themes/exploit/docs/28478.pdf
[14] Tyler Bletsch, Xuxian Jiang, Vince W.Freeh(2011). Jump-Oriented Programming: A New Class of Code-Reuse Attack.
[15] Mathias Payer, Thomas R. Gross(2013). String-Oriented Programming: When ASLR is not Enough.
[16] SECURE COMPUTER SYSTEMS “Blind Return-Oriented Programming(BROP)”, http://www.scs.stanford.edu/brop/

表 1．各種攻撃と防御手法の関係

		スタック オーバーフロー	Return-to-libc	Off-by-one	GOT 書き換え	Return-to - register	ヒープ オーバーフロー	Heap-spraying	Return-Oriented Programing	Jump-Oriented Programming	String-Oriented Programming	Blind Return-Oriented Programming
OS の防御機能	データ実行防止機能	○		○		○	○	○				
	ASLR	○	○		○		○					
	ASCII-Armor											
gcc コンパイラの防御機能	SSP	○	○	○		○			○			
	PIE		○						○	○	○	
	RELRO				○							