メモリ破損脆弱性に対する攻撃の調査と分類

鈴木 舞音† 上原 崇史† 金子 洋平† 堀 洋輔† 馬場 隆彰‡ 齋藤 孝道‡

†明治大学大学院，‡明治大学
214-8571 神奈川県川崎市多摩区東三田 1-1-1
{ce36028, ce36006, ce36017, ce46029, ee17033}@meiji.ac.jp, saito@cs.meiji.ac.jp

あらまし ソフトウエアのメモリ破損脆弱性を悪用する攻撃，いわゆる，メモリ破損攻撃が次々と登場し問題となっている．メモリ破損攻撃とは，メモリ破損脆弱性のあるプログラムの制御フローを攻撃者の意図する動作に変えることである．一般的には，Buffer Overflow攻撃とも呼ばれている．OSやコンパイラでの防御・攻撃緩和機能が開発されてはいるが，それらを回避する更なる攻撃も登場している．そこで，本論文では，メモリ破損脆弱性の分類を行い，制御フローを不正に書き換えるまでの攻撃に関して調査し，分類する．

A Survey of Attacks against Memory Corruption Vulnerabilities

Maine SUZUKI† Takafumi UEHARA† Yohei KANEKO† Yosuke HORI†
Takaaki BABA‡ Takamichi SAITO‡

†Graduate School of Meiji University, ‡Meiji University
1-1-1, Higashimita, Tama-ku Kawasaki-shi, Kanagawa, 214-8571, Japan
{ce36028, ce36006, ce36017, ce46029, ee17033}@meiji.ac.jp, saito@cs.meiji.ac.jp

Abstract It has become a serious problem that the number of attacks that exploits memory corruption vulnerability in software is increased. Protection/mitigation technologies against it in OS or with compilers have been developed, but further attacks to avoid them have been appeared. In this paper, we survey to classify the memory corruption vulnerabilities, and their attacks

1. はじめに

脆弱性の種類を識別するための共通の脆弱性タイプの一覧である CWE (Common Weakness Enumeration) [1]において，メモリ破損脆弱性の中に分類される，いわゆる Buffer Overflow は，現在でも NVD (National Vulnerability Database) [2]での報告が絶えない脆弱性の一つである．

Buffer Overflow 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，1988 年に発生した Morris Worm [4]をはじめ，2014 年に発生した Internet Explorer の Use After Free 脆弱性 [5]など，現在に至るまで，様々な攻撃に悪用されている．メモリ破損脆弱性を利用した攻撃手法に対して，OS やコンパイラでの防御・攻撃緩和機能（以降，対策技術という）が開発されてはいるが，様々な攻撃手法を組み合わせることによってそれらを回避する更なる攻撃も登場している．

そこで，本論文では，攻撃者によって悪用されるメモリ破損脆弱性の分類を行い，制御フローを不正に書き換えるまでの攻撃に関して調査し，分類する．また，脆弱性の分類は CWE (Version 2.8) に従う．

本論文では，32bit の Linux OS における C 言語で書かれたプログラムかつ GCC (GNU Compiler Collection) でコンパイルされた Linux ELF (Executable and Linkable Format) 形式[6]のバイナリに焦点を絞る．

1 メモリ破損脆弱性

メモリ破損脆弱性 (CWE-119) [7]は，プログラム内で確保されているバッファの境界外への読み書きが可能な際に発生する．メモリ破損脆弱性を利用して，攻撃者は，意図する制御フローへの書き換え，任意の情報の読み出し，または，システムの破壊が可能となる．この脆弱性を利用した攻撃をメモリ破損攻撃という．

1.1 Buffer Overflow

Buffer Overflow (CWE-120) [8]とは，入力データを検査しないプログラムにおいて，プログラム内でバッファとして確保している範囲を超えて，メモリ領域にデータが書き込まれてしまう現象及び脆弱性のことである．メモリ破損脆弱性 (CWE-119) の一つでもある．この脆弱性を利用した攻撃を Buffer Overflow 攻撃という．攻撃者はバッファサイズ以上のデータでバッファを溢れさせ，プログラムの制御フローを攻撃者の意図した動作に変えるようにメモリの内容を書き換える．

1.1.1 Stack-based Buffer Overflow

Stack-based Buffer Overflow (CWE-121) [9]とは，スタック領域に確保されたバッファで Buffer Overflow を引き起こす脆弱性である．スタック領域では，関数ポインタやフレームポインタ，リターンアドレスが書き換え対象となる．

1.1.2 Heap-based Buffer Overflow

Heap-based Buffer Overflow (CWE-122) [10]とは，ヒープ領域に確保されたバッファで Buffer Overflow を引き起こす脆弱性である．ヒープ領域では，ポインタが書き換え対象となる．

1.1.3 BSS-based Buffer Overflow

BSS[11]領域は，プログラム実行時に自動的に生成される領域で，静的変数や大域変数のうち初期化されていないものやこれらの変数に 0 が代入されているものが入る．

BSS-based Buffer Overflow とは，BSS 領域に確保されたバッファで Buffer Overflow を引き起こす脆弱性である．BSS 領域での攻撃では，関数ポインタが書き換え対象となる．これに該当する CWE の分類はない．

2 メモリ破損脆弱性と併用される脆弱性

ここでは，攻撃時，メモリ破損と併用して利用される脆弱性について説明する．Buffer Overflow 攻撃において，メモリ破損脆弱性だけでなく，書式文字列の問題や数値処理の問題を併用することで，攻撃者は，対策技術を回避することがある．ここで，CWE の分類によると書式文字列の問題 (CWE-134) や数値処理の問題 (CWE-189) は，メモリ破損脆弱性に含まれないことに注意されたい．

2.1 書式文字列の問題

書式文字列の問題 (Uncontrolled Format String, CWE-134)[12]とは，書式指定子のない printf 系列の関数を不正利用し，メモリ上の任意の値を読み書きできてしまう脆弱性のことである．例えば，printf(input)関数呼出しが不正に使用される場合，input 変数への入力文字列が書式指定子として使用されてしまう．

2.2 数値処理の問題

数値処理の問題 (Numeric Errors, CWE-189) [13]とは，不適切な数値演算または数値変換に関する脆弱性のことである．数値処理の問題に分類される Integer Overflow or Wraparound (CWE-190) [14]が，特に Buffer Overflow 攻撃と併用して利用される脆弱性である．Integer Overflow or Wraparound は，演算結果の値が，演算式の型で表現できる範囲を超える場合に発生する脆弱性である．

3　メモリ破損攻撃

ここでは，バッファの確保されている領域別にメモリ破損攻撃手法について説明する．

3.1　Stack-based Buffer Overflow 攻撃

Stack-based Buffer Overflow 攻撃とは，Stack-based Buffer Overflow (CWE-121) により，関数へのポインタやフレームポインタ (saved %ebp)，及び，リターンアドレス (saved %eip) を“不正な命令コード (shellcode) の先頭アドレス”で書き換えなどをする攻撃のことである．図 1 は，main 関数から func 関数を呼び出した正常時の関数のスタックフレームを示す．

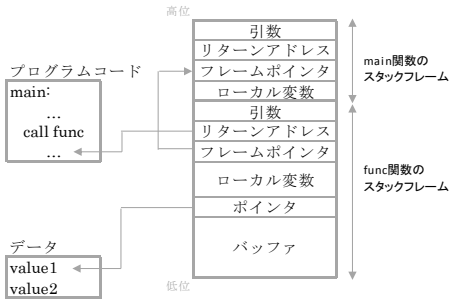

図 1　正常時のスタックレイアウト

3.1.1　リターンアドレス書き換え攻撃

リターンアドレス書き換え攻撃とは，関数終了後に次に実行すべきプログラムコードへのアドレスであるリターンアドレスを“不正な命令コード (shellcode) の先頭アドレス”で書き換える攻撃のことである (図 2 参照)．

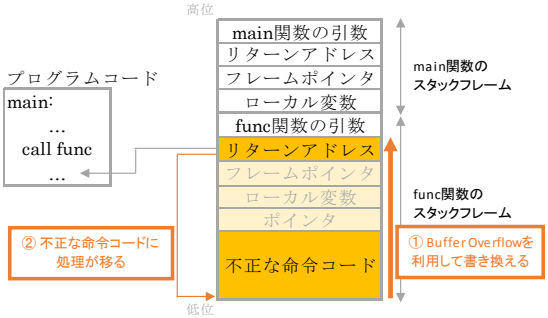

図 2　リターンアドレス書き換え攻撃時のスタックメモリイメージ

3.1.1.1　Return-to-libc 攻撃

Return-to-libc 攻撃[15]とは，リターンアドレス書き換え攻撃の一種で，リターンアドレスを“悪用するライブラリ関数へのアドレス”で書き換え，その関数呼出しに必要な引数をスタックフレームに用意することで，攻撃者の意図したライブラリ関数を呼び出す攻撃のことである．Return-to-libc 攻撃は，Exec-shield に実装されているコード実行防止機能を回避することが可能となる[17][18]．

3.1.1.2　Return-to-plt 攻撃／Return-to-strcpy 攻撃

Return-to-plt 攻撃[16]とは，リターンアドレス書き換え攻撃の一種で，ライブラリ関数を呼び出す際に参照する間接ジャンプテーブルである“PLT (Procedure Linkage Table) 領域 (.plt セクション) へのアドレス”でリターンアドレスを書き換える攻撃のことである．また，その関数呼出しに必要な引数をスタックフレームに用意することで，攻撃者の意図したライブラリ関数を呼び出すことができる．

Return-to-plt 攻撃の一種である Return-to-strcpy 攻撃[16]とは，リターンアドレスを strcpy@plt へのアドレスで書き換え，strcpy 関数に必要なスタックフレームを用意することで，strcpy 関数を呼び出す攻撃のことである．NULL 文字 (¥0x00) を含むアドレスを任意のメモリ領域に書き込む場合に利用する手法である．

このように.plt セクションを利用する攻撃手法は，ASCII-armor[17]を回避することが可能となる．

3.1.1.3　Return-to-Register 攻撃

Return-to-Register 攻撃[19]とは，ret 命令実行後にレジスタが指しているアドレスに不正な命令コードを挿入し，その上で，“そのレジスタ値に実行を移す命令群が格納されているアドレス”でリターンアドレスを書き換える攻撃のことである．

図 3 は，esp レジスタを利用した場合の攻撃手法を示す．main 関数の ret 命令実行後には，esp レジスタは引数を指している．そこで，引数とリターンアドレスを，“不正な命令コード”

及び“jmp %esp に対応するバイト列”のいずれもが格納されているアドレスで書き換える．main 関数の ret 命令実行時に，書き換えられたリターンアドレスである“jmp %esp に対応するバイト列”が格納されているアドレスを eip レジスタに pop する．すると，jmp %esp により，esp レジスタが指している不正な命令コードに実行が移る．

また，jmp %esp 以外にも，call %esp, push %esp, ret, call eax などの命令も悪用できる．

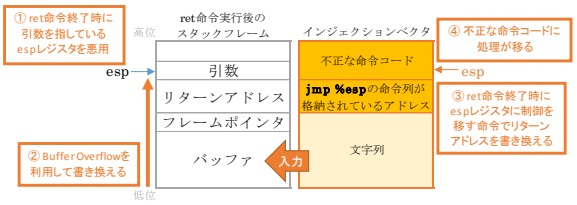

図 3　Return-to-Register 攻撃時のスタックメモリイメージ

Return-to-Register 攻撃は，ASLR (Address Space Layout Randomization，アドレス空間配置のランダム化)[17][18]を回避することが可能となる．

3.1.1.4 Return-Oriented Programming 攻撃

ROP (Return-Oriented Programming) 攻撃 [20]とは，ASLR によって配置アドレスがランダム化されないライブラリ関数や実行プログラムの命令コード (以降，ガジェットという) を組み合わせて shellcode の代替とする攻撃のことである．

ROP 攻撃の場合，ret 命令で終わる命令コード (以降，ROP ガジェットという) を組み合わせる．ROP 攻撃の他に，jmp 命令で終わる命令コードを使用する JOP (Jump-Oriented Programming) 攻撃[21]や ROP ガジェットと JOP ガジェットを使い分け，書式文字列の問題を利用する SOP (String-Oriented Programming) 攻撃[22]もある．

ROP 攻撃や JOP 攻撃は，データ領域におけるコード実行防止機能と ASLR を回避することが可能となる．更に，SOP 攻撃では，コード実行防止機能と ASLR，SSP (Stack Smashing Protector) [17][18][23]を回避することが可能となる．

3.1.2　フレームポインタ書き換え攻撃

フレームポインタ書き換え攻撃とは，不正な命令コード，偽のフレームポインタ，及び，偽のリターンアドレスを含めた偽のスタックフレームでバッファを溢れさせ，フレームポインタが“偽のスタックフレーム”を指すように書き換える攻撃のことである．この攻撃の対象プログラムには，少なくとも二つ以上の関数を必要とする (図 4 参照)．

実行の流れを以下に示す．ここで，func 関数は main 関数から呼び出されるとする．

(A) func 関数終了間際の leave 命令で，攻撃者により書き換えられた func 関数のフレームポインタ (偽のフレームポインタのアドレス) が ebp レジスタに pop される．この時，ebp レジスタは偽のフレームポインタを指す．

(B) func 関数の ret 命令で，main 関数に復帰する

(C) main 関数終了間際の leave 命令で，mov %ebp, %esp が実行され，ebp レジスタの値 (偽のフレームポインタのアドレス) が esp レジスタにコピーされる．さらに，偽のフレームポインタ (不正な命令コードのアドレス) が ebp レジスタに pop される．この時，ebp レジスタが不正な命令コードの先頭を指し，esp レジスタが偽のリターンアドレスを指す．

(D) main 関数の ret 命令で，偽のリターンアドレス (不正な命令コードのアドレス) が eip レジスタに pop され，不正な命令コードに実行が移る．

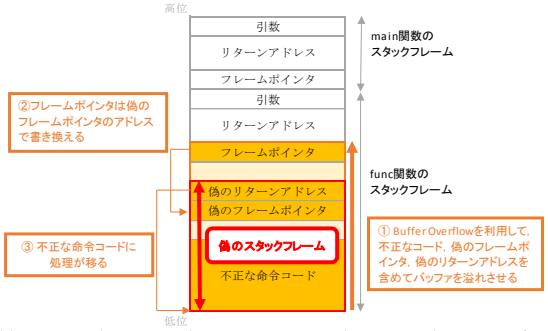

図 4　フレームポインタ書き換え攻撃時のスタックメモリイメージ

3.1.3 Off-by-one 攻撃

Off-by-one 攻撃[24]とは，Off-by-one Error (CWE-193) [25]のあるプログラムを利用し，フレームポインタ書き換え攻撃と同様の手順で行われる攻撃のことである．ただし，フレームポインタの下位 1 バイトだけしか書き換えられないという制約がある．ここで，CWE の分類によると Off-by-one Error は，メモリ破損脆弱性に含まれず，数値処理の問題（CWE-189）に含まれることに注意されたい．

3.2 Heap-based Buffer Overflow 攻撃

Heap-based Buffer Overflow 攻撃とは，Heap-based Buffer Overflow (CWE-122) により，関数ポインタやポインタを“不正な命令コード（shellcode）の先頭アドレス”で書き換える攻撃のことである[18]．また，ASLR 回避やアドレスの正確な位置が不明な際の攻撃を目的として，不正な命令コードをヒープ領域に大量に書き込み，攻撃の成功率を高める Heap Spray [26]という手法もある．

二回以上同じメモリ領域を解放してしまう脆弱性（CWE-415）[27]を利用して，Heap-based Buffer Overflow 攻撃を行う Double Free 攻撃[26]もある．また，未使用のメモリへの参照が残っている脆弱性 (CWE-416) [28]を利用し，その参照先を不正な命令コードを参照する様に書き換える Use After Free 攻撃[26]もある．

3.3 BSS-based Buffer Overflow 攻撃

BSS-based Buffer Overflow 攻撃とは，BSS 領域に確保されたバッファを溢れさせて，より高位にある関数ポインタの値を“不正な命令コード（shellcode）の先頭アドレス”で書き換える攻撃のことである[19]．書き換え手法は，GOT 書き換え攻撃（3.4.1 節）と同様である．

3.4 その他の攻撃

ここでは，スタック，ヒープ及び BSS 領域ではないメモリ破損脆弱性を悪用した攻撃について説明する．

3.4.1 GOT 書き換え攻撃

GOT（Global Offset Table）書き換え攻撃[19][26]とは，共有ライブラリ関数を呼び出す際に参照する間接アドレステーブルである GOT 領域（.got.plt または, .got セクション）を不正な命令コードへジャンプするようにテーブルの内容を書き換える攻撃のことである．主に，関数ポインタが悪用される．

図 5 は遅延バインド有効時の printf 関数呼び出しのイメージを示す．

(a) printf 関数が呼び出されると，printf@plt を参照する
(b) 次に，.got.plt セクションのテーブル内容を参照する．
(c) 初回関数呼び出し時には，.plt セクションに戻る．
(d) 二回目以降の関数呼び出し時には，直接共有ライブラリ関数へジャンプする．

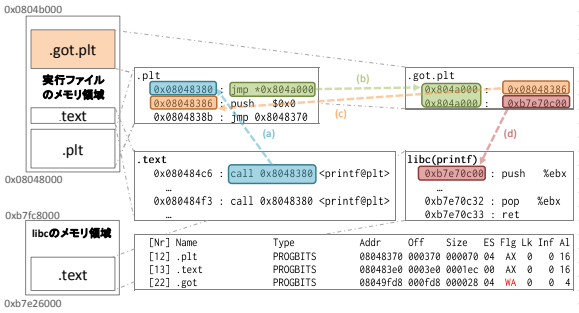

図 5 遅延バインド有効時の printf 関数呼び出しのイメージ

図 6 は遅延バインド無効時の printf 関数呼び出しのイメージを示す．

(A) printf 関数が呼び出されると，printf@plt を参照する．
(B) 次に，.got.plt セクションのテーブル内容を参照する．
(C) 直接共有ライブラリ関数へジャンプする．

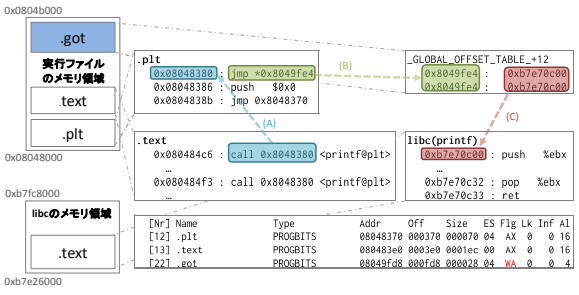

図 6 遅延バインド無効時の printf 関数呼び出しのイメージ

GOT 書き換え攻撃において，遅延バインド有効時は，.got.plt セクションのテーブル内容を“不正な命令コード（shellcode）の先頭アドレス”に書き換える．つまり，図 5 の (c) または (d) のジャンプ先を“不正な命令コード (shellcode) の先頭アドレス”に書き換える．しかし，遅延バインド無効時は，.got セクションは読み取り専用のため，書き換えることはできない．

3.4.2 Stack Pivoting

Stack Pivoting[26]とは，関数終了時，関数呼び出し時，または，ポインタ変数呼び出し時にレジスタが指しているアドレスに不正な命令コードを挿入し，その上で，“そのレジスタ値に実行を移す ROP ガジェット”（表 1 参照）でリターンアドレス，GOT テーブル，または，ポインタ変数を書き換える攻撃手法のことである（図 7 参照）．よって，Return-to-Register 攻撃もこの手法に分類される．

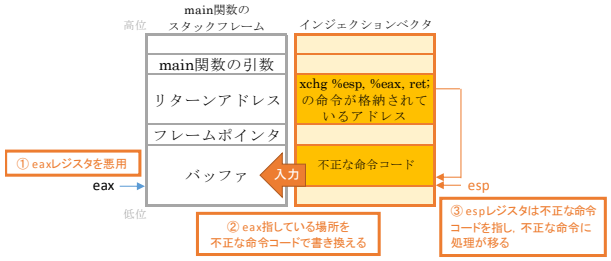

図 7　eax レジスタを悪用した際の Stack Pivoting のスタックメモリイメージ

表 1　Stack Pivoting 用 ROP ガジェット一覧

ROP ガジェット	説明
xchg %esp, %eax, ret	esp レジスタの値と eax レジスタの値を交換する．
mov %esp, %eax, ret	esp レジスタの値を eax レジスタの値にコピーする．
add %esp, [値], ret	esp レジスタに[値]分加える．

4　メモリ破損脆弱性と併用される攻撃

ここでは，メモリ破損脆弱性と併用される攻撃や攻撃手法について説明する．

4.1　書式文字列攻撃

書式文字列攻撃[19]とは，メモリ上の任意の値を読み書きできてしまう脆弱性（CWE-134）を利用し，実行中のプログラムのメモリに不正な命令コードを送り込む，任意のメモリアドレスを読み出すなどを目的とする攻撃のことである．主に，ASLR 回避のために任意のアドレスを計算するために用いられる．

4.2　整数オーバーフロー攻撃

整数オーバーフロー攻撃[19]とは，整数演算の結果が，数値を扱う変数が取り扱える範囲を超えてしまう脆弱性（CWE-189，CWE-190）を利用する攻撃のことである．

4.3　Improper Null Termination

Improper Null Termination[29]とは，strncpy 関数のコピー元の文字列のサイズよりも strncpy 関数で指定したバッファサイズのほうが小さい場合，コピー先の文字列が NULL 文字（¥0x00）で終端されない脆弱性（CWE-170）を利用して，主に SSP により挿入された canary 値を読み出す際に利用することである．ここで，Improper Null Termination は，脆弱性（CWE-170）と攻撃手法のいずれも指す用語であることに注意されたい．読みだした canary 値を含めて，バッファの内容を書き換えることで，関数終了時に SSP により行われる canary 値の検査を回避することが可能となる．

5　対策技術を回避する攻撃手法

ここで，対策技術を回避する攻撃手法について説明する．

5.1　Brute Force 攻撃

ここでの Brute Force 攻撃とは，攻撃に利用する任意のメモリアドレスと一致するまで繰り返し攻撃を試みる攻撃のことである．SSP や ASLR を回避するために用いられる手法である．特に，canary 値に対して行われることが多い[26]．1 バイトずつバッファを溢れさせて canary 値と一致するかどうかを試行する byte-by-byte [30]という手法もある．

5.2 Canary 偽造攻撃

Canary 偽装攻撃とは，canary 値を偽装することによって，Stack-based Buffer Overflow Buffer Overflow 攻撃を実現する攻撃のことである．Linux (Ubuntu14.04 Kernel 3.13.0-34-generic, gcc-4.9.1) は，プロセスが生成された時点での canary を使いまわしているということを我々は確認した．よって，例えば，親プロセスで生成された canary を何らかの方法で取得できれば，子プロセスでは，canary を偽造できる可能性がある．

6 メモリ破損攻撃の分類

ここで，本論文で紹介したメモリ破損攻撃の分類を行う（表 2 参照）．

7 まとめ

本論文では，メモリ破損脆弱性や，メモリ破損脆弱性とその他の脆弱性を分類し，それらを併用した攻撃手法の分類を行った．

参考文献

[1] CWE: Common Weakness Enumeration, http://cwe.mitre.org/index.html

[2] NVD: 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] Vulnerability Summary for CVE-2014-1776, http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-1776

[6] The Linux ELF HOWTO, http://cs.mipt.ru/docs/comp/eng/os/linux/howto/howto_english/elf/elf-howto.html#toc1

[7] CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer, http://cwe.mitre.org/data/definitions/119.html

[8] CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow'), http://cwe.mitre.org/data/definitions/120.html

[9] CWE-121: Stack-based Buffer Overflow, http://cwe.mitre.org/data/definitions/121.html

[10] CWE-122: Heap-based Buffer Overflow, http://cwe.mitre.org/data/definitions/122.html

[11] Stevens, W. Richard (1992). Advanced Programming in the Unix Environment. Addison–Wesley. Section 7.6., http://bluetechs.files.wordpress.com/2014/03/advanced-programming-in-the-unix-environment-by-w-richard-stevens-stephen-a-rago-ii-edition.pdf

[12] CWE-134: Uncontrolled Format String, http://cwe.mitre.org/data/definitions/134.html

[13] CWE-189: Numeric Errors, http://cwe.mitre.org/data/definitions/189.html

[14] CWE-190: Integer Overflow or Wraparound, http://cwe.mitre.org/data/definitions/190.html

[15] Infosec Writers,"Bypassing non-executable-stack during exploitation using return-to-libc", http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf

[16] Linux exploit development part 4 - ASCII armor bypass + return-to-plt, http://ihazomgsecurityskillz.blogspot.jp/2011/05/linux-exploit-development-part-4-ascii.html

[17] 角田 佳史, 金子 洋平, 鈴木 舞音, 上原 崇史, 齋藤 孝道, バッファオーバーフロー攻撃に関する防御技術の調査, 2014 年, FIT 情報科学技術フォーラム

[18] 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)

[19] Müller, Tilo. "ASLR smack & laugh reference." Seminar on Advanced Exploitation Techniques. 2008.

[20] Erik Buchanan, Ryan Roemer, Hovav Shacham, and Stefan Savage. When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC. In Proceedings of the 15th ACM Conference on Computer and Communications Security (CSS) ,October 2008.

[21] T. Bletch, X. Jiang, V. X. Freeh, and Z. Liang, "Jump-oriented programming: a new class of code-reuse attack", In Proceedings of the 6th ACM Conference on Computer and Communications Security (CSS), 2011, pp. 30-40

[22] Payer, Mathias, and Thomas R. Gross. "String oriented programming: when ASLR is not enough." Proceedings of the 2nd ACM SIGPLAN Program Protection and Reverse Engineering Workshop. ACM, 2013.

[23] オープンソース・ソフトウェアのセキュリティ確保に関する調査報告書, 第Ⅲ部, セキュアな実行コードの生成・実行環境技術に関する調査, https://www.ipa.go.jp/files/000013695.pdf

[24] Vallentin, Matthias. "On the evolution of buffer overflows." Munich, May (2007).

[25] CWE-193: Off-by-one Error, http://cwe.mitre.org/data/definitions/193.html

[26] Röttger, Stephen. "Malicious Code Execution Prevention through Function Pointer Protection." (2013).

[27] CWE-415: Double Free, http://cwe.mitre.org/data/definitions/415.html

[28] CWE-416: Use After Free, http://cwe.mitre.org/data/definitions/416.html

[29] CWE-170: Improper Null Termination, http://cwe.mitre.org/data/definitions/170.html

[30] Seredinschi, Dragoş-Adrian, and Adrian Sterca. "ENHANCING THE STACK SMASHING PROTECTION IN THE GCC." Studia Universitatis Babes-Bolyai, Informatica 56.4 (2011).

表 2 メモリ破損脆弱性を利用した攻撃の分類

	攻撃名 (攻撃)	書き換え対象		書き換え内容
		メモリ領域 (領域)	対象	
1	リターンアドレス書き換え	スタック	リターンアドレス	shellcode
2	Return to Register			ROPガジェット
3	Return-Oriented Programming			.text (libc)
4	Return to libc			.plt ※
5	Return to plt			.plt (strcpy関数) ※
6	Return to strcpy			.got.plt (GOT Overwrite参照)
8	Off-by-one		フレームポインタ (下位1byteだけ)	shellcode
				ROPガジェット
				.text (libc)
				.plt ※
9	フレームポインタ書き換え		フレームポインタ	shellcode
				ROPガジェット
				.text (libc)
				.plt ※
			関数ポインタ	.got.plt (GOT Overwrite参照)
		.bss		
		ヒープ		
10	Double Free		ポインタ	shellcode
11	Use After Free			ROPガジェット
				.text (libc)
				.plt ※
12	GOT Overwrite	.got.plt	間接アドレス	shellcode
				ROPガジェット
				.text (libc)
				.plt ※

備考: ※ さらなる攻撃の手段として利用される．.plt に strcpy 関数を利用した攻撃も含める．