Vol.2023-DPS-194 No.24
Vol.2023-CSEC-100 No.24
2023/3/6

様々なCPUアーキテクチャにおける ROP攻撃可能性の検証

多木優馬 [1, a)]　福光正幸 [2, b)]　湯村翼 [1, c)]

概要：Return Oriented Programming(ROP)攻撃は，プログラム内の命令片を利用し，任意の処理を行わせる攻撃手法であり，様々な派生手法も登場している．ROP攻撃への対策として，高い計算能力を持つCPUを活用するControl-Flow Integrity(CFI)や，広いアドレス空間を活用するAddress Space Layout Randomization(ASLR)といったセキュリティ機構が存在する．ただし，組込み機器では，プロセッサに対する制約によってこれらのセキュリティ機構を実装できない場合もある．本研究では，組込み機器に対するROP攻撃の可能性を調査するため，x86，ARM32，ARM64環境に対してROP攻撃検証を行った．プロセッサエミュレータQEMUを用いて攻撃対象環境を構築し，そこで稼働する脆弱なテストプログラムに対してROP攻撃を実施した．検証の結果，x86環境とARM32環境へのROP攻撃は成立し，ARM64環境へのROP攻撃は不成立であった．この結果について考察し，関数呼び出し方法を工夫することでリターンアドレスの書き換えを緩和できることを明らかにした．

キーワード：セキュリティ，組込み機器，ROP攻撃，CPUアーキテクチャ

1. はじめに

IoT 機器の普及によって組込み機器の需要が増加している. それに伴い, IoT機器を悪用した脅威も発生している．実際，IoT機器を狙ったマルウェアであるMiraiは，ルータやWebカメラといった約50万台のIoT機器を乗っ取った事例が存在する．さらに，AmazonやTwitterといった著名なWebサービスに対し, 乗っ取ったIoT機器を使ってDDoS攻撃を仕掛け，アクセス困難な状況に陥れた [1]．また，プログラム内の命令片を利用し，任意の処理を行わせる攻撃手法であるReturn Oriented Programming (ROP)攻撃 [2]と，その派生技術も数多く存在する．さらに，ROP攻撃はその汎用性の高さから，組込み機器に応用される恐れがある．

以上のことから, 本研究では組込み機器へのROP攻撃対策に着目する．その対策の一例として，不正な制御フローを検知するセキュリティ機構である Control-Flow Integrity (CFI)の実装 [3]などが進んでいる．CFI は，組込み機器向けの ARMv8-M アーキテクチャに対応した実装例がある [4]．しかし，組込み機器は種類が多く，ROP対策が施されていない機器が存在する可能性がある．

このため，本研究では様々なアーキテクチャ対するROP攻撃の可能性を調査する．CPUをQEMU [5]を用いてエミュレートして攻撃対象の環境を構築し，そこで稼働するROP攻撃検証用のテストプログラムに対して ROP攻撃を試みる．

なお，x86環境，ARM32環境における ROP攻撃の可能性については報告済みである [6]．本稿では x86 環境とARM32環境の更なる詳細, ARM64環境への ROP攻撃結果とその考察について報告する．

2. ROP攻撃

ROP攻撃について説明するにあたり, ROP攻撃で利用されるバッファオーバーフロー攻撃 [7]と，ROP攻撃が考案されたきっかけとなったNX bit [7]について述べる．

2.1 バッファオーバーフロー (BOF)攻撃

BOF 攻撃 [7]はバッファ境界チェックを行わない入力関数を狙った攻撃手法である．この攻撃を利用し，リターンアドレスを書き換えることでプログラムの制御を奪うことができる(図 1)．さらに，リターンアドレスでジャンプさせるアドレスを，シェルコードと呼ばれる攻撃者が作成した攻撃コードにジャンプさせることで，任意の処理の実行が可能となる．

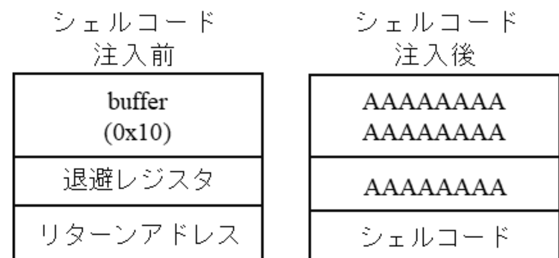

図 1　バッファオーバーフローによる
シェルコード埋め込みの概要

2.2 No eXecutable bit (NX bit)

NX bit [7]はバッファオーバーフロー脆弱性を悪用したシェルコード埋め込みを阻止するためのセキュリティ機構である．指定した領域以外でのコード実行を禁止することで，スタック領域内やヒープ領域内でのシェルコードの実行を防ぐことができる．

2.3 Return Oriented Programming (ROP)攻撃

ROP攻撃 [2]とは，ガジェットと呼ばれるROP攻撃に利

1 北海道情報大学
Hokkaido Information University Nishi-Nopporo 59-2, Ebetsu, Hokkaido 069-8585, Japan
2 長崎県立大学
University of Nagasaki 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki 851-2195, Japan
a) turkey0727@outlook.jp
b) fukumitsu@sun.ac.jp
c) yumu@yumulab.org

ⓒ2023 Information Processing Society of Japan

用できるアセンブラコードの命令片を繋ぎ合わせて ROP Chain を構築し，任意の処理を行わせることができる攻撃である(図 2)．攻撃者が作成したシェルコードではなく，プログラムにもともと存在する実行許可されたコードを利用するため，NX bit を回避することができる．また，様々なアーキテクチャに応用することも可能で，x86 アーキテクチャだけではなく，ARMv8-A アーキテクチャなどへの攻撃にも応用されている [8]．

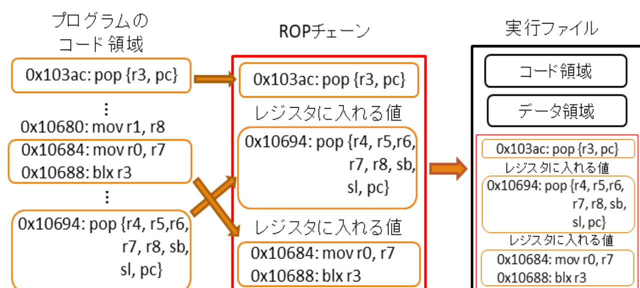

図 2　BOF 攻撃によるシェルコード埋め込みの概要

3. ROP 攻撃に有効なセキュリティ機構

ROP 攻撃対策として有効なセキュリティ機構の代表例は以下の通りである．

3.1 Stack Smashing Protector (SSP)

SSP [9]とは，スタック領域に push されたリターンアドレスの直前にカナリアという乱数値を挿入し，関数の実行終了時にカナリアが変更されているかを確認することで，BOF 攻撃を検知するセキュリティ機構である．これにより，BOF 攻撃によるリターンアドレスの書き換えが困難になり，ひいては ROP 攻撃対策に繋がる．ただし，ヌル文字に関するバグである Improper Null Termination により，カナリアを特定し，この対策を回避する手法も存在する．

3.2 Address Space Layout Randomization (ASLR)

ASLR [10]とは，プログラムの再配置時に，プログラムのコード領域やデータ領域の配置先をランダム化し，特定のアドレスへのアクセスを困難にするセキュリティ機構である．これにより，ROP 攻撃による命令片の利用が困難になる．ただし，ROP 攻撃の派生技術である Just-In-Time ROP(JIT-ROP) [11]の場合，アドレスのリークが可能になることもある．また，アドレス幅が 32bit 以下の環境では，エントロピーが不足し，総当たり攻撃によるアドレスリークも可能となる．このため，アドレス幅が小さい組込み機器への ASLR 実装は効果的ではない．

3.3 Control-Flow Integrity (CFI)

CFI [3]とは，プログラム実行時に制御フローを監視し，正常な制御フローと比較することで異常な制御フローを検知するセキュリティ機構である．これにより，ROP 攻撃による命令片への不正なジャンプを検知することができる．しかし，CFI 実装は負荷が大きいため，低性能なプロセッサへの実装は困難である．

4. ROP 検証の詳細

4.1 QEMU

ROP 攻撃の検証にはプロセッサエミュレータ QEMU [5]を使用した．QEMU は CPU やメモリといったハードウェアをエミュレートするプロセッサエミュレータである．QEMU を使うことで，任意のタイミングでゲスト環境のレジスタの値やメモリの内容を確認することができる．検証に利用するホストとゲスト間の構成図は図 3 のようになっている．

また，本稿ではテストプログラムへの攻撃を行っているが，ゲスト OS は QEMU の設定によりホスト OS 以外との通信を遮断しており，外部からの攻撃を受けないようになっている．また，ゲスト環境は自身が構築したものであり，攻撃を行っても外部に影響を及ぼさないよう配慮している．

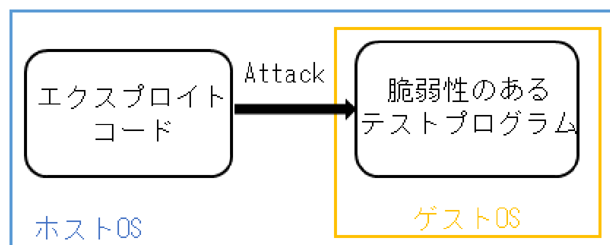

図 3　ROP 攻撃検証環境の構成

4.2 対象とするアーキテクチャと OS の組み合わせ

本稿の ROP 攻撃検証対象として，i686 で稼働する CentOS6 環境（以降，x86 環境と呼ぶ)と，Arm Cortex-A53 の AArch32 で稼働する Raspberry Pi OS 環境（以降，ARM32 環境と呼ぶ)，そして Arm Cortex-A53 の AArch64 で稼働する Nuttx 環境(以降，ARM64 環境と呼ぶ)を採用して構築した（表 1)．

表 1 攻撃対象の環境

検証対象	プロセッサ	アーキテクチャ	OS	実行ステート
x86環境	i686	x86	CentOS6	-
ARM32環境	Arm Cortex-A53	Armv8-A	RaspiOS	AArch32
ARM64環境	Arm Cortex-A53	Armv8-A	Nuttx	AArch64

i686 プロセッサは，x86 アーキテクチャを採用した，主に PC で使われるプロセッサである．i686 で稼働する CentOS6 環境を構築した理由は，i686 プロセッサで採用されている x86 アーキテクチャも組込み機器で利用されることがあるということと，PC と組込み機器とのセキュリティ機構の違いについての調査のためである．ARM 32 環境を構築した理由は，組込み機器向けのプロセッサとしては高性能な ARM Cortex-A53 プロセッサについて事前調査することで，

ⓒ2023 Information Processing Society of Japan

今後行う予定である低性能なプロセッサを用いた組込み機器への ROP 攻撃検証の比較を行うためである．ARM64 環境を採用した理由は ARM32 環境との違いについて調査するためである．

4.3 検証に用いる攻撃対象のテストプログラムについて

x86 環境と ARM32 環境での検証用テストプログラムには，文献 [12]の ROP 実験用テストプログラムを使用している．ARM32 環境上におけるテストプログラムの正常な動作結果は図 4 の通りになる．このプログラムの入力関数はバッファ境界チェックを行わない gets 関数を用いており，BOF 攻撃が可能となっている．また，ARM64 環境においては本研究にて自作したテストプログラムを使用している．このプログラムにも BOF 脆弱性が存在し，バッファサイズを超える入力が可能となっている．

なお，本研究ではアーキテクチャの違いによる ROP 攻撃の可能性を調査するため，ASLR や SSP といったコンパイラや OS 側で実装されるセキュリティ機構は無効化している．

```
pi@raspberrypi:~/ROP_test$ ./checkname
This program is for Raspi
Input name: yuuma
NG
pi@raspberrypi:~/ROP_test$ ./checkname
This program is for Raspi
Input name: Mike
OK
```

図 4 テストプログラムが正常に実行された際のターミナル

4.4 対象とする ROP 攻撃

ROP 攻撃には様々な派生技術が存在するが，本研究では単純な ROP 攻撃のみを用いて検証を行う．この理由としては，単純な ROP 攻撃であっても制御を奪われるような脆弱な組込み機器の調査をすることが本研究の目的であるためである．ROP 攻撃の検証にあたってエクスプロイトコードの記述は Python3.9.13 を使って行い，ROP Chain はエクスプロイト開発ライブラリである Pwntools [13]を用いて構築した．また，ROP Chain 構築に必要な命令片の探索は rp++ [14]，ropper[15]を用いた．

5. x86 環境への ROP 攻撃

5.1 x86 での関数呼び出し

x86 アーキテクチャでの関数呼び出しは，call 命令によって行われる．この call 命令は call 命令の次の命令のアドレスをリターンアドレスとしてスタックに push し，指定した関数のアドレスにジャンプする．呼び出された関数側(callee)では，スタックフレームのベースアドレスを指す EBP を push した後，スタックトップのアドレスを指す ESP を減算して callee で扱うバッファを確保する（図 5）．関数の最後では，EBP を pop した後，ret 命令によってリターンアドレスにジャンプして関数呼び出し側(caller)に戻る．

5.2 x86 でのリターンアドレスの書き換え

バッファからリターンアドレスまでのオフセットバイトは， EBP – BOF 攻撃対象のバッファのアドレス ＋ スタックベースポインタのサイズとなる．なお，callee で EBP をベースとしたアドレス変換を行わない場合は関数プロローグ時に EBP を push しないが，その場合は，リターンアドレスまでのオフセットバイトは EBP – BOF 攻撃対象のバッファのアドレスとなる．

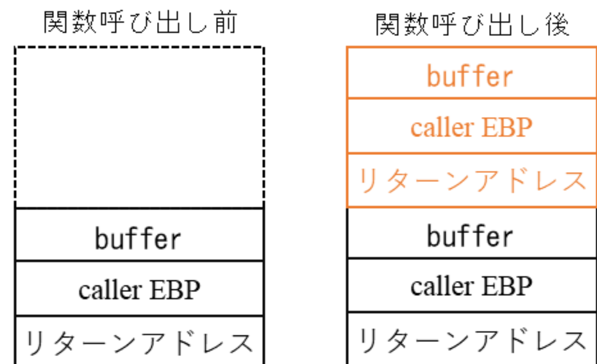

図 5 関数呼び出し時のスタックの動作

5.3 ROP 攻撃に必要な命令片

x86 環境で動作するテストプログラムへの ROP 攻撃検証にあたり，使用した命令は mov 命令と，pop 命令，そして int 0x80 命令の 3 つである．本実験での ROP 攻撃の検証では，mov 命令を使って文字列/bin/sh を BSS セクションのある領域に格納し，pop 命令を使用して EAX レジスタに 11，EBX レジスタに文字列/bin/sh へのアドレスをそれぞれ格納し，最後に int 0x80 命令を実行することで execve(/bin/sh)を呼び出してコンピュータの制御を奪うことを目的としている．

5.4 ROP チェーンの構築と送信

作成したエクスプロイトコードの内，ROP Chain の構築を行っている箇所は図 6 の通りである．検証の結果，攻撃は成立し，コンピュータの制御を奪うことができた．ここでは，ホスト環境から cat /etc/redhat-release を実行してゲスト環境の OS のバージョンを表示することで，攻撃が成立したことを確認できた(図 7).

ⓒ2023 Information Processing Society of Japan

```
def attack(conn, **kwargs):
    payload = b'A' * (0x1c)          # offset
    payload += pop_dcb_addr          # pop {edx, ecx, ebx}
    payload += bss_addr              # edx = bss_addr
    payload += b'/bin'               # ecx = "/bin"
    payload += p32(0)                # ebx = 0

    payload += mov_edx_ecx_addr      # *bss_addr = "/bin"
    payload += pop_dcb_addr          # pop {edx, ecx, ebx}
    payload += bss_addr_offset       # edx = bss_addr + 4
    payload += b'/sh\0'              # ecx = "/sh\0"
    payload += p32(0)                # ebx = 0

    payload += mov_edx_ecx_addr      # *(bss_addr + 4) = "/sh\0"
    payload += pop_ebp_addr          # pop ebp
    payload += p32(11)               # eax = 11
    payload += pop_dcb_addr          # pop {edx, ecx, ebx}
    payload += p32(0)                # edx = 0
    payload += p32(0)                # ecx = 0
    payload += bss_addr              # ebx = bss_addr
    payload += int80_addr            # int 0x80
```

図 6　ROP Chain を構築するエクスプロイトコード (x86 環境)

```
[*] Switching to interactive mode
$
[DEBUG] Sent 0x1 bytes:
    10 * 0x1
[DEBUG] Received 0x3 bytes:
    b'NG\n'
NG
$ cat /etc/redhat-release
[DEBUG] Sent 0x18 bytes:
    b'cat /etc/redhat-release\n'
[DEBUG] Received 0x21 bytes:
    b'CentOS Linux release 6.0 (Final)\n'
CentOS Linux release 6.0 (Final)
```

図 7　エクスプロイトコードによる ROP 攻撃実行例 (x86 環境)，OS バージョン確認コマンド cat /etc/redhat-release の実行結果

6. ARM32 環境への ROP 攻撃

6.1 ARM32 での関数呼び出し

ARM32 アーキテクチャでの関数呼び出しも，call 命令で行われるが, ARM32 の call 命令はリターンアドレスの push は必須ではない．リターンアドレスを格納するリンクレジスタ(LR)が存在するため，callee が他の関数を呼び出さない関数(リーフ関数)だった場合，リターンアドレスの push は行われない. callee が他の関数を呼び出す関数 (非リーフ関数)だった場合, x86 同様リターンアドレスの push が行われる．非リーフ関数の場合，基本的に callee の関数プロローグ時に push {fp, lr}命令があり，LR，FP (フレームポインタレジスタ)の順でスタックに push される. その後に, callee 側で使用するバッファを用意する. 関数の最後には pop {fp, lr}によって，FP，LR の順で pop した後，ret 命令でリターンアドレスへジャンプする．

6.2 ARM32 でのリターンアドレスの書き換え

callee が非リーフ関数だった場合，リターンアドレスの書き換えが可能である．リターンアドレスまでのオフセットバイトは i686 同様，SP – BOF 攻撃対象のバッファのアドレス + フレームポインタのサイズを加算した値となる．

6.3 ROP 攻撃に必要な命令片

ARM32 環境で動作するテストプログラムへの ROP 攻撃検証にあたり，使用した命令は，pop 命令のみである．本検証の ROP 攻撃では pop 命令を使って R0 レジスタに文字列/bin/sh へのアドレスを格納し，プログラムカウンタを指す PC レジスタに system 関数のアドレスを格納することで system(/bin/sh)を呼び出してコンピュータの制御を奪うことを目的としている．

6.4 ROP チェーンの構築と送信

作成したエクスプロイトコードの内，ROP Chain の構築を行っている箇所は図 8 の通りである．検証の結果，攻撃は成立し，コンピュータの制御を奪うことができた．ここではホスト環境側から lsb_release -a を実行してゲスト環境の OS のバージョンを表示することで攻撃が成立したことを確認できた(図 9).

```
def attack(conn, **kwargs):
    payload = b'A' * (0x10 + 0x4)    # name + saved register
    payload += pop_r0_addr           # pop {r0, r4, pc}
    payload += binsh_addr            # r0 = "/bin/sh" address
    payload += b"ABCD"               # r4 = junk data
    payload += system_addr           # pc = sysytem() address
```

図 8　ROP Chain を構築するエクスプロイトコード (ARM32 環境)

```
[*] Switching to interactive mode
$
[DEBUG] Sent 0x1 bytes:
    b'\n'
[DEBUG] Received 0x3 bytes:
    b'NG\n'
NG
$ lsb_release -a
[DEBUG] Sent 0xf bytes:
    b'lsb_release -a\n'
[DEBUG] Received 0x1e bytes:
    b'No LSB modules are available.\n'
No LSB modules are available.
[DEBUG] Received 0x62 bytes:
    b'Distributor ID:\tRaspbian\n'
    b'Description:\tRaspbian GNU/Linux 10 (buster)\n'
    b'Release:\t10\n'
    b'Codename:\tbuster\n'
Distributor ID:    Raspbian
Description:    Raspbian GNU/Linux 10 (buster)
Release:    10
Codename:    buster
```

図 9 エクスプロイトコードによる ROP 攻撃実行例 (ARM32 環境)，OS バージョン確認コマンド lsb_release -a の実行結果

7. ARM64 環境への ROP 攻撃

ARM64 環境に，ROP 攻撃検証を行った．結果，攻撃は成立しなかった.原因は BOF 攻撃でのリターンアドレスの書き換えができなかったためである．本節では，ARM64

ⓒ2023 Information Processing Society of Japan

アーキテクチャでの関数呼び出し規約についてと，デバッガによる調査結果をまとめる．

7.1 ARM64 アーキテクチャでの関数呼び出し

ARM64 環境では ARM32 同様，callee がリターンアドレスをスタックに push するが，push の前にバッファを用意することができ，リターンアドレスをバッファよりも下位アドレスに配置することが可能である．本 ROP 攻撃検証で使用したテストプログラムでは上記の条件が当てはまり，callee のバッファを上書きしてもリターンアドレスの書き換えができない状態になっていた．(図 10)

図 10　ARM64 環境での buffer1 に対する BOF 攻撃時のスタック領域

7.2 デバッグによる調査

GDB デバッガを用いて，攻撃コード送信時の ARM64 環境のスタック領域を確認した．図 11 から分かる通り，リターンアドレスがバッファよりも下位アドレスに配置され，リターンアドレスを書き換えられていないことがわかる．

図 11　攻撃コード実行時の ARM64 環境のスタック領域，オレンジ枠で囲っている箇所がバッファ，青枠で囲っている箇所がリターンアドレス

8. 考察

8.1 x86 環境と ARM32 環境での ROP 攻撃対策

今回，ROP 攻撃が成立した x86 環境と ARM32 環境においても，OS やコンパイラが実装しているセキュリティ機構を用いれば，ROP 攻撃の緩和が可能となる．ただし，ASLR と SSP については 3 節で述べた通り，環境によっては回避が可能となっている．ROP 攻撃対策の代表例として CFI が挙げられる．一般に，CFI が適切に実装された場合の回避は非常に困難であるといわれているため，CFI の実装が重要であると言える．

8.2 ARM64 環境への ROP 攻撃可能性

ARM64 環境での ROP 攻撃の失敗は，リターンアドレスが書き換えられないことを原因として挙げていた．ただし，リターンアドレスの書き換えが可能であった場合においても ROP 攻撃は困難であると考えられる．本小節では，ROP 攻撃を困難にするアーキテクチャ設計とセキュリティ機構について述べる．

8.2.1 固定命令長

x86 とは異なり，Arm アーキテクチャの命令の長さは固定である．これにより，命令片のもととなる機械語をずらして使用し，意図しない命令を利用するという手法が利用できないため，利用できる命令片の数が大きく減少する．

8.2.2 PC レジスタの書き換え禁止

AArch32 では，pop pc 命令を利用し，PC レジスタに任意の値に書き込むことができた．しかし，AArch64 では，ジャンプ命令，ret 命令以外を用いた PC レジスタの書き換えが禁止された．これにより，命令片の終端がジャンプ命令か ret 命令のどちらかに制限され，利用できる命令片が減少する．

8.2.3 PAC の実装

ARM64 での代表的な ROP 攻撃対策として Armv8.3 から実装された PAC(Pointer Authentication Code) [16]が挙げられる．PAC は関数ポインタの先頭 16bit に埋め込まれる暗号化された値であり，関数ポインタに埋め込まれる認証コードであり，ジャンプ命令実行直前に目的のポインタの PAC を確認することで，不正なジャンプを検知し，例外を起こしてプログラムを保護することができる．これにより，ROP 攻撃による不正なジャンプ命令が困難になる．

8.3 他のアーキテクチャへの応用

これまでの考察結果の通り，リターンアドレスを格納できる固有のレジスタを設置し，リターンアドレスの push を callee に委ねることができるアーキテクチャ設計にすることで，BOF 攻撃回避が実現できると考えられる．この対策について，従来の ASLR や SSP などのセキュリティ機構に比べ，関数呼び出し方法の工夫のみで実現できるため，低リソースなアーキテクチャにも実現しやすいと考えられる．

9. おわりに

本稿では，x86 環境と ARM32 環境，そして ARM64 環境における ROP 攻撃の検証を行った．本稿で行った ROP 攻撃のデモでは ASLR や SSP といった OS やコンパイラ側で実装されるセキュリティ機構を無効にし，プロセッサの違

いによる ROP 攻撃の可能性について調査した．その結果，x86 環境と ARM32 環境に対する ROP 攻撃成立と，ARM64 環境に対する ROP 攻撃の不成立を確認した (表 2)．この原因を考察し，関数呼び出しの工夫のみで BOF 攻撃によるリターンアドレスの書き換えを緩和できることが分かった．今後，さらに多くのアーキテクチャ，特に低スペックな組込み機器で使用されるアーキテクチャに対しての ROP 攻撃検証を進める．

表 2 ROP 攻撃検証結果，LR はリンクレジスタ，PC はプログラムカウンタレジスタ

検証対象	OS	アーキテクチャ設計の違い			ROP攻撃成立可否
		LRの有無	命令長	PCの直接書き換え可否	
x86環境	CentOS6	×	可変長	×	○
ARM32環境	RaspiOS	○	固定長	○	○
ARM64環境	Nuttx	○	固定長	×	×

謝辞　国立研究開発法人情報通信研究機構主催の SecHack365 にて，トレーナーの坂井弘亮氏よりご教示いただいたプログラムの動作原理とデバッグ手法の知識が本研究を実地するうえでの礎となりました．坂井氏ならびに SecHack365 関係者の皆様に感謝申し上げます．

参考文献

[1] セキュリティ通信: IoT デバイスを標的にした「Mirai」の攻撃手口から対策まで徹底解説, available from <https://securitynews.so-net.ne.jp/topics/sec_20129.html> (accessed 2023-02-09)
[2] Ryan, R., Erik, B., Hovav, S., Stefan, S.: Return-oriented programming: Systems, languages, and applications, ACM Transactions on Information and System Security (TISSEC),Vol. 15, No.1, pp. 1—34 (2012).
[3] Martin, A., Mihai, B., UIfar, E., Jay, L.: Control-flow integrity principles, implementations, and applications, ACM Transactions on Information and System Security (TISSEC), Vol. 13, No.1, pp. 1—40 (2009).
[4] 河田智明，本田晋也，松原豊，高田広章: Arm TrustZone for Armv8-M を利用したマルチタスク対応 CFI の検討，組込みシステムシンポジウム 2018 論文集, pp. 71−74,(2018).
[5] QEMU: , available from <https://www.qemu.org/> (accessed 2023-02-09).
[6] 多木優馬，福光正幸，湯村翼:組込み機器への ROP 攻撃の検証のためのプロセッサエミュレータを用いた調査，情報処理北海道シンポジウム 2022 論文集,pp.185-188,(2022).
[7] 品川高廣: NX bit を回避するバッファオーバーフロー攻撃の防止手法，情報処理学会研究報告コンピュータセキュリティ (CSEC), 2006-CSEC-034, pp. 215−222, (2006).
[8] 赵利军，李民，彭诚: ARMv8-A Return-Oriented Programming 实现方法. 计算机应用与软件, Vol. 11, (2018).
[9] IPA 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, available from < https://www.ipa.go.jp/files/000013695.pdf > (accessed 2023-02-09)
[10] PaX, T.: PaX address space layout randomization (ASLR), available from <http://pax. grsecurity. net/docs/aslr. txt>, (2003).
[11] Snow, K.Z., Monrose, F., Davi, L., Dmitrienko, A., Liebchen, C., Sadeghi, A.R., Just-in-time CodeReuse: On the effectiveness of fine-grained address space layout randomization, IEEE Symposium on Security and Privacy(S&P), (2013).
[12] kozos.jp　ROP 実験用サンプル, available from <http://kozos.jp/samples/rop-sample.html> (accessed 2023-02-09)
[13] Pwntools: , available from <https://github.com/Gallopsled/pwntools> (accessed 2023-02-09)
[14] rp++: , available from <https://github.com/0vercl0k/rp> (accessed 2023-02-09)
[15] ropper: , available from <https://github.com/sashs/Ropper> (accessed 2023-02-09)
[16] ARM pointer authentication: , available from <https://lwn.net/Articles/718888/> (accessed 2023-02-09)

ⓒ2023 Information Processing Society of Japan