Vol.2020-DPS-182 No.28
Vol.2020-CSEC-88 No.28
2020/3/12

# IoTの安全性向上のためのOSセキュリティ機能調査

松下 意悟[1]　鄭 俊俊[1]　藤松 由里恵[2]　金井 遵[2]　毛利 公一[1]

**概要**：急速なIoT化が進み，インターネットに接続する組込み機器であるIoT機器が増加していき，これらの機器は，高機能化，小型化して おり，多くのセンシティブな情報を取り扱うようになってきている．それに伴い，IoT機器を対象とする攻撃も増えてきており，IoT機器固有の脆弱性を突く攻撃の割合も増えてきている．IoT機器は，非力で機能の少ないCPUとOSを搭載しているものもあり，Intel 64 アーキテクチャ CPU 上で動くWindowsやLinuxに比べてソフトウェアの耐タンパ性は低く，様々な攻撃の対象となりやすい．この問題を解決するためには，IoT機器に既存のセキュリティ機能よりさらに有効に働くものが必要である．以上の背景から，IoT機器のCPUとして採用が広がっているARM Cortex-M シリーズ上で動作するOSで有効に働くセキュリティ機能を明らかにするために，Linuxディストリビューションの1つであるUbuntuとARM Cortex-M シリーズのCPU上で動作するOSのセキュリティ機能についての調査を行い，IoT機器上で動作するOSが搭載しているセキュリティ機能の実装状況を明らかにした．

## 1. はじめに

近年，PCやスマートフォン以外にも，あらゆるものをインターネットに接続させ，それぞれが相互に通信を行うことにより新たなサービスを提供する，IoT(Internet of things) への取り組みが広まっている．2020年には，日本でも低遅延，高信頼性，多数同時接続を特徴とする5G通信のサービスが開始され，これまで以上に急速なIoT化が進むと予想される．また，IoT機器は，高機能化，小型化しており，多くのセンシティブな情報を取り扱うようになってきている．

それに伴い，IoT機器を対象とする攻撃も増えてきており，IoT機器固有の脆弱性を突く攻撃も増えてきている [1]．2016年にはMiraiと呼ばれるマルウェアがIoT機器を乗っ取り，Botnetを形成することにより大規模なDDoS攻撃が発生した [2]．日本においても，IoT機器のセキュリティを確保することが重視され，総務省と国立研究開発法人情報通信研究機構がISPと連携し，IoT機器のセキュリティの調査「NOTICE(National Operation Towards IoT Clean Environment)」を2019年2月20日から開始している [3]．IoT機器は，非力で機能の少ないCPUとOSを搭載しており，Intel®[a] 64 アーキテクチャ CPU 上で動くWindows®[b] やLinux®[c] に比べてソフトウェアの耐タンパ性は低く，様々な攻撃の対象となりやすい．この問題を解決するためには，性能の低いIoT機器に既存のセキュリティ機能よりさらに有効に働くものが必要である．

本論文では，IoT機器のCPUとして採用が広がっているARM®[d] Cortex® -M シリーズ CPU 上で動作するOSに必要なセキュリティ機能を明らかにし，既存のセキュリティ機能の知見を得るために，以下を行った．

- IoT機器に利用されるOSには，どのようなものがあり，どのようなセキュリティ機能が搭載されているか，また，どのような実現方法が取られているかの調査
- Linuxディストリビューションの1つであるUbuntu®[e]が搭載しているセキュリティ機能にどのようなものがあり，それらのセキュリティ機能は，Kernel，GCC，glibcなど，どのようなソフトウェアで実装されているかの調査

以下，本論文では，2章でARM Cortex-Mシリーズの機能について述べ，3章でIoTに利用されるOSと実装されているセキュリティ機能ついて述べる．4章では，Ubuntuが搭載しているセキュリティ機能について述べ，5章で，

[1] 立命館大学
Ritsumeikan University
[2] 株式会社 東芝
Toshiba Corporation

[a] Intel は，アメリカ合衆国およびその他の国における Intel Corporation またはその子会社の商標または登録商標です．

[b] Windows は，米国 Microsoft Corporation の米国およびその他の国における登録商標です．

[c] Linux は，Linus Torvalds 氏の米国，日本およびその他の国における登録商標または商標です．

[d] ARM，Cortex は，ARM Limited（またはその子会社）の EU およびその他の国における登録商標もしくは商標です．

[e] Ubuntu は，Canonical Ltd. の商標または登録商標です．
その他本稿に掲載の商品，機能等の名称は，それぞれ各社が商標として使用している場合があります．

ⓒ 2020 Information Processing Society of Japan

IoT 機器のセキュリティ機能の現状をまとめ，IoT 機器のセキュリティを向上させるために，Ubuntu のセキュリティ機能をどのように見るべきか述べ，5 章でまとめる．

## 2. ARM Cortex-M シリーズ

IoT 機器に採用される OS のセキュリティ機能の実装に利用されている ARM Cortex-M シリーズの機能である，特権レベル，メモリ保護，スタックポインタについてそれぞれ述べる．本論文では，Cortex-M シリーズの命令セットアーキテクチャとして特に ARMv7-M と ARMv8-M を対象とし，以下では Cortex-M と記す．調査には，それぞれ ARMv7-M Architecture Reference Manual[4] と ARMv8-M Architecture Reference Manual[5] を参照した．

### CPU の特権レベル

Cortex-M では，権限分離として Privilege Levels を提供しており，2 つの動作モードと 2 つの特権レベルが存在する (図 1)．特権モードはカーネルの実行に使用され，非特権モードではアプリケーションの実行などユーザモードに使用される．動作モードはスレッドモードとハンドラモードが存在する．スレッドモードでは通常動作を行い，ハンドラモードでは例外発生時のハンドラ実行時の動作を行う．例外発生時はカーネルが対処を行うようにするため，ハンドラモードは特権モードにのみ存在する．

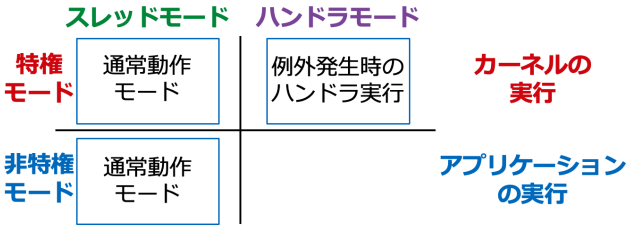



図 1　Cortex-M の動作モードと特権モード

### MPU(Memroy Protection Unit)

Cortex-M は他の一般的な CPU である Intel 64 アーキテクチャ CPU に搭載されている MMU(Memroy Management Unit) が存在しない．しかし，メモリ保護を提供する MPU(Memory Protection Unit) の搭載がマイコンベンダによって選択可能となっている．MPU は，最大 16 個のメモリ領域に対して，特権レベルに応じた，読み取り専用，読み書き可能，アクセス不可，実行不可の属性を与えることができる．

### スタックポインタ

Cortex-M では，スタックポインタが特権レベルごとに存在していおり，特権モードでは MSP(Main Stack Pointer) を使用し，非特権モードでは PSP(Process Stack Pointer) を使用する．また，それぞれのスタックポインタに対して，下限値を保持するスタックリミットレジスタも存在し，MSP の下限値は MSPLIM に，PSP の下限値は PSPLIM に格納される．プログラム実行中に，MSP または PSP がそれぞれのスタックリミットレジスタに格納されている値を下回った場合，CPU が Usage Fault を発生させる．

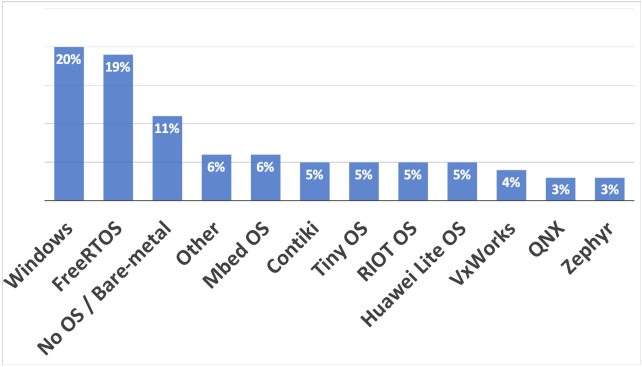



図 2　IoT 機器で採用される OS のシェア

## 3. IoT 機器向け OS のセキュリティ機能

IoT 機器に採用される OS のセキュリティを向上させるために，これらの OS のセキュリティ機能の現状を明らかにする必要がある．本章では，Cortex-M 上で動作する OS にどのようなセキュリティ機能が搭載されているか，ドキュメントを調査した内容について述べる．さらに，それらのうちオープンソースである OS について，どのように実現されているのかソースコードから，実装の確認を行った結果について述べる．

### 3.1 ドキュメント調査

Eclipse® Foundation が調査した，IoT 機器に使用される OS のシェア [6] を元に調査を行った．ただし，[6] では Linux が除かれている．その内訳を図 2 に示す．図 2 に示されたもののうち Cortex-M 上で動作することがドキュメントに明記されている OS に加え，国内で広く採用される T-Kernel 系の OS のうち Cortex-M 上で動作する eT-Kernel Compact，カーネルコンフィグで MMU を無効にした Linux を対象として各 OS のドキュメントからセキュリティ機能の存在の調査を行った．以下でそれぞれの OS が持つセキュリティ機能を挙げる．

**FreeRTOS™**

- スタックオーバフローの検知
- 非特権モードでは，タスクが持つスタックと最大 3 つのユーザ定義可能なメモリ領域にしかアクセスできない．

**Mbed™OS**

ドキュメントには明記されていない．

**RIOT OS**

スタックオーバフローからの保護

**Huawei® Lite OS**

ドキュメントには明記されていない．

**VxWorks®**

- Non-Executable pages: ページ単位でコードの実行/実行不可の制御
- Stack guard page: スタックによる他のメモリ領域の侵害防止
- コードとリードオンリーデータの保護
- Kernel page table isolation(KPTI): カーネルとユーザ空間で別のページテーブルを持つ
- SecureBoot，TPM，ARM TrustZone® のサポート

**Zephyr®**

- メモリ分離：メモリの所有者に基づく属性割当て
- スタック保護：スタックオーバフローの検知
- スレッド分離：特権モード時と非特権モード時の環境の分離

**eT-Kernel Compact**

- リングプロテクションの提供
- メモリ保護機能

**Linux Kernel**

ドキュメントなどは存在せず，不明．

### 3.2 実装調査

前節で述べた OS のうち，セキュリティ機能が明記されているもの，オープンソースであるものに関して，搭載されているセキュリティ機能がどのように実装されているかをソースコードから調査を行なった．その調査結果について述べる．

#### 3.2.1 FreeRTOS

FreeRTOS の実装を確認する調査において対象とするソースコードは，FreeRTOS の公式サイト [7] から取得した v10.2.1 である．

FreeRTOS のセキュリティ機能は，以下に示す 2 つが挙げられる．

- スタックオーバフローの検知
- 非特権モードにおけるメモリアクセス制限

以下でこの 2 つについて述べる．

**スタックオーバフローの検知**

FreeRTOS において，スタックオーバフローの検知方法は 2 種類ある．

(1) スタックポインタの位置が，スタック領域の先頭より，低位アドレスであるか比較する

(2) スタックの最後 16 バイトが既知の値か比較する

これらのスタックオーバフローの検知は，コンテキストスイッチ時に行われる．

まずは，(1) の方法について述べる．スタック領域の通常時とスタックオーバフロー発生時のメモリの様子を図 3 に示す．(1) の方法では，スタック領域の開始アドレスと，現在のスタックポインタのアドレスを比較することにより，スタックオーバフローが，発生しているかを判断している．

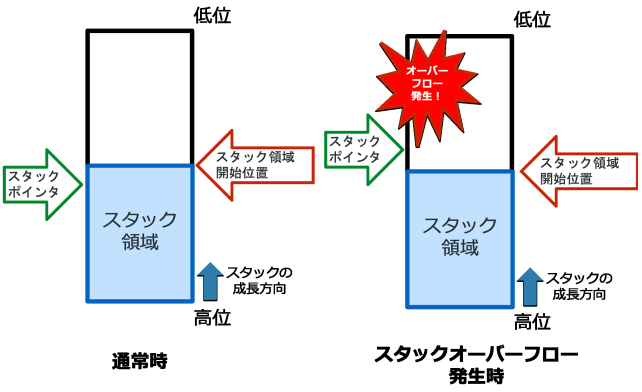



図 3 FreeRTOS のスタックオーバフロー検知方法 (1) が有効な時のスタック領域

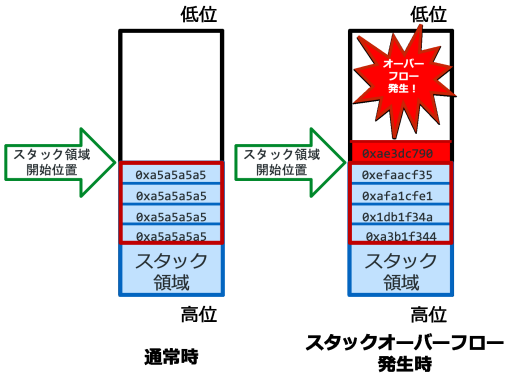



図 4 FreeRTOS のスタックオーバフロー検知方法 (2) が有効な時のスタック領域

次に，(2) の方法について述べる．スタック領域の通常時とスタックオーバフロー発生時のメモリの様子を，図 4 に示す．(2) の方法は，スタック領域の開始アドレスが既知の値か比較する．スタックオーバフローが発生したときは，スタック領域外の領域が使用されるため，スタック領域の端の値を既知の値にしておき，その値が既知ではない別の値になっていればスタックオーバフローが発生したと判断する．

**非特権モードにおけるメモリアクセス制限**

FreeRTOS において，MPU を搭載している場合，プロセスの作成に xTaskCreateRestricted 関数の使用が推奨される．xTaskCreateRestricted 関数はプロセス作成の際に以下のメモリ領域に関する制限がかけられる．

(1) 非特権モードでは，タスクがもつスタックにしかアクセスできない．

(2) 非特権モードでは，ユーザ定義可能な 3 つのメモリ領域にしかアクセスできない．

ⓒ 2020 Information Processing Society of Japan

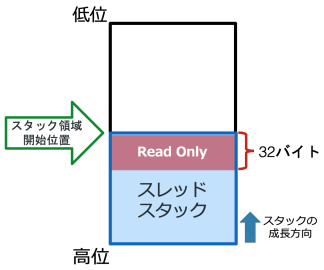



図 5　RIOT OS のスタックオーバフローを保護する仕組み

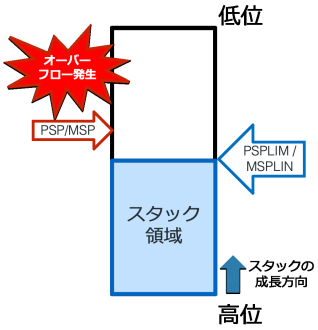



図 6　Zephyr における PSPLIM・MSPLIM を用いたスタックオーバフローの検知

まずは，(1) について述べる．(1) の機能は，xTaskCreateRestricted 関数内部で呼ばれる，vPortStoreTaskMPUSettings 関数により，スタック領域に対して以下の属性を付与することで実装がされている．

- 非共有メモリ
- 特権モードで RW 可・非特権モードで RW 可能
- 実行不可

FreeRTOS では，MPU の設定が行われていない領域については，特権モード時でのみアクセスが可能となるよう設定されているため，非特権モード時はタスクが持つスタック領域のみにアクセスが可能となる．

次に (2) について述べる．(2) も (1) と同様に，xTaskCreateRestricted 関数内部で呼ばれる，vPortStoreTaskMPUSettings 関数によって実装が行われている．vPortStoreTaskMPUSettings 関数内で xTaskCreateRestricted 関数の引数の一つである，作成するプロセスの情報を持つ pxTaskDefinition 構造体のメンバ変数のうち，メモリ領域の情報を持つ構造体である xRegions 変数の情報を基に最大 3 つのメモリ領域において，以下の属性をメモリ領域に割り当てることが可能となっている．

- 読み取り専用または読み書き可能
- 実行不可属性

また，xRegions 変数内で定義されているメモリ領域に非共有メモリ属性が必ず付与される．

### 3.2.2　RIOT OS

RIOT OS の実装を確認する調査において対象とするソースコードは，RIOT OS の GitHub® ページの Release-2019.07[8] から取得したものである．

RIOT OS のセキュリティ機能には，「スタックオーバフローからの保護」が存在する．RIOT OS では，スタックオーバフローの検知には MPU を利用している．その動作を**図 5** に示す．RIOT OS では，スタック領域の先頭アドレスから低位アドレス側に向かって 32 バイトの領域を MPU により読み取り専用属性を付与することにより，スタックポインタがこの領域に達し，書き込みを行うと，Memory Management Fault が発生する．それによって，スタックオーバフローの検知が可能となる．

### 3.2.3　Zephyr

Zephyr の実装を確認する調査において対象とするソースコードは，Zephyr の GitHub ページ [9] から取得したバージョン 2.0.0 のものである．

Zephyr のセキュリティ機能には，「メモリ分離」，「スタック保護」，「スレッド分離」の 3 つがある．それぞれについて述べる．

**メモリ分離**

メモリ分離は，メモリの所有者に基づく属性割り当てを行うものである．ユーザモードでスレッドを動作させる時，スレッドのスタック領域を特権モードで読み書き可能，非特権モードで読み書き可能になるように設定を行なっている．また，MPU によってメモリ領域に対して属性の設定を行っていない場合，特権レベルのみ読み書き可能となっている．

**スタック保護**

スタック保護は，スタックオーバフローの検出を行うものである．Zephyr では 2 つの方法でスタックオーバフローの検知が可能である．

- PSPLIM・MSPLIM を使用する方法
- MPU を使用する方法

これら 2 つの方法を同時に使用することはできなく，どちらか片方のみ有効にすることができる．それぞれの方法について述べる．

PSPLIM・MSPLIM を使用する方法では，コンテキストスイッチ時に PSPLIM または MSPLIM を設定することにより，PSP または MSP が，下限値を超えてアクセスをする時に Usage Fault が発生し，スタックオーバフローを検知することが可能となる．この時の様子を**図 6** に示す．

MPU を使用する方法では，スタック領域の開始アドレスから 32 バイト下方の領域を読み取り専用とする．この時のメモリの様子を**図 7** に示す．RIOT OS と同様に，読み取り専用領域であるスタックガード領域を作成し，スタックポインタが，スタックガード領域に書き込みを行うと，Memory Management Fault が発生するため，スタックオーバフローの発生を検知することができる．RIOT OS と異なる点は，スタック領域から 32 バイト下方にスタックガード領域の開始アドレスを設定する点である．

ⓒ 2020 Information Processing Society of Japan

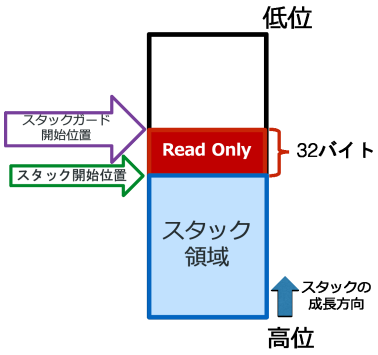



図 7　MPU を用いてスタックガード領域を作成する様子




図 8　スレッド分離が有効なスタック

**スレッド分離**

スレッド分離は，特権モードと非特権モードにおいて，環境を分離する機能である．例を挙げると，特権モード時のスタックと非特権モード時のスタックを隔離するために特権モードで読み取り専用，非特権モードでアクセス不可となる属性のスタックガード領域を設るといった機能が実装されている．この時のスタックの様子を図 8 に示す．

## 4. Linux におけるセキュリティ機能

Cortex-M 上で動作する OS に必要なセキュリティ機能を検討するにあたり，既存の汎用的なシステムのセキュリティ機能にどのようなものがあるのか知見を得る必要がある．そのため，汎用システムのオープンソースである OS として Linux のディストリビューションの 1 つである Ubuntu のセキュリティ機能を調査した．調査にあたり，Ubuntu Wiki の Ubuntu が搭載しているセキュリティ機能を掲載している Web サイト [10] を参考にし，それぞれのセキュリティ機能が，Kernel，GCC，glibc などどのソフトウェアで実現されているかを分類し，さらにその機能について調査した．本章では，その調査結果について述べる．

### 4.1 カーネルにおける実装

**SYN cookies**

TCP 通信確立時の 3way-handshake において，クライアント側が SYN パケットを大量に送りつけ，ディスティネーション側のリソースを枯渇させる SYN Flood 攻撃に対する防御機能であり，SYN Flood 攻撃を検知した場合に，クライアントからの ACK パケットを受け取るまでコネクション情報を持たないようにする．

**Kernel Livepatch**

再起動不要なカーネルアップデート方法の提供を行う．kernel/livepatch/Kconfig の CONFIG_LIVEPATCH で有効無効を切替可能である．

**Filesystem Capabilities**

拡張ファイル属性のサポートし，SELinux などでファイルの詳細なアクセス制御などを行えるようにする．

**Non-Executable Memory**

ヒープ領域やスタック領域などに実行不可属性を付与する．

**Stack ASLR**

スタック領域の配置場所のアドレスランダム化を行う．

**Libs/mmap ASLR**

動的ライブラリのロード位置のアドレスランダム化を行う．

**Exec ASLR**

コード領域の配置場所のアドレスランダム化を行う．

**brk ASLR**

ヒープ領域の配置場所のアドレスランダム化を行う．

**VDSO ASLR**

VDSO(Virtual Dynamic Shared Object) 領域の場所のアドレスランダム化を行う．

**/proc/$pid/maps protection**

ASLR が有効である場合に，プロセスのメモリマップは攻撃者にとって重要な情報となる．そのため，/proc/<プロセスの PID>/maps が他のユーザから読み取られてはならない．そこで，ファイルを読み取り専用にし，このファイルへの読み取りアクセスが発生した場合に，ptrace_may_attach() を用いたチェックを行うことで，プロセス自身，またはプロセスの所有者以外の maps ファイルへの読み取りアクセスを不可にしている．

**Symlink restrictions**

シンボリックリンクによって，TOCTOU(Time of check to time of use) が原因で引き起こされる想定していないファイルへの編集をできないようにする．

**Hardlink restrictions**

ハードリンクによって，TOCTOU(Time of check to time of use) が原因で引き起こされる想定していないファイルへの編集をできないようにする．

**FIFO restrictions**

FIFO ファイルを所有者以外が O_CREATE モードで open できないようにする．

**Regular file restrictions**

Regular ファイルを所有者以外が O_CREATE モードで open できないようにする．

**ptrace scope**

ptrace できるプロセスをデバッガの子孫プロセスに制限

ⓒ 2020 Information Processing Society of Japan

する．

**0-address protection**

NULL ポインタが指す位置が 0 番地であることを悪用し，mmap システムコールにて作成した仮想メモリ空間の 0 番地に配置した任意のコードを特権レベルで実行する NULL アドレスデリファレンス攻撃の防御機能．mmap システムコールでマッピング可能な最小アドレスを 64k 番地にすることにより NULL デリファレン攻撃を防ぐ．

**/dev/mem protection**

物理メモリ (/dev/mem) へのアクセスを拒否する．DOS エミュレータの dosemu やウインドウシステムである X の古いものは，/dev/mem 経由で IO メモリへアクセスするため，CONFIG_IO_STRICT_DEVMEM の値を n にすることで，/dev/mem 経由で IO メモリへのアクセスが可能となる．

**/dev/kmem disabled**

カーネルの仮想メモリへのインターフェースである/dev/kmem の削除．

**Block module loading**

カーネルルートキットのインストールを防ぐためにカーネルモジュールのロードを不可にする．sysctl コマンドで modules_disabled の値を 1 にすることにより本機能が有効となる．

**Read-only data sections**

カーネルデータセクションへ読み取り専用属性を付与する．

**Stack protector**

カーネル空間のスタック領域保護機能．スタック領域中にカナリア領域を作成することにより，ローカル変数のオーバフローによる値の書き換えを防ぐ．CONFIG_CC_STACKPROTECTOR の値を y にすることにより，GCC の”-fstack-protection”オプションを有効にして，カーネルをビルドする．

**Module RO/NX**

カーネルモジュールのデータセクションへ読み取り専用，または実行不可属性を付与する．

**Kernel Address Display Restriction**

boot/vmlinuz*，/boot/System．map*，/sys/kernel/debug/，/proc/slabinfo を root 権限でのみ読み取れるようにし，カーネルアドレスの秘匿化を行う．

**Kernel Address Space Layout Randomisation**

カーネル空間における ASLR の実装である．kASLR では以下についてランダム化を行う．

- カーネルイメージをロードする物理メモリのベースアドレス
- カーネルイメージをロードする仮想メモリのベースアドレス
- カーネルモジュールのロードするメモリのベースアドレス
- カーネルスタックのベースアドレス
- ダイナミックメモリのベースアドレス

**Blacklist Rare Protocols**

脆弱性が含まれている可能性のある，滅多に使用されないプロトコルのカーネルモジュールの自動ロードを行わない．

**Block kexec**

管理者によって sysctl を使用して kexec の無効化を可能とし，kexec を自由に使用できないようにする．

**UEFI Secure Boot (amd64)**

UEFI の機能の 1 つである，起動時に実行するプログラムが改ざんされていないかを検証するセキュアブートへの対応．

**TPM(Trusted Platform Module)**

TCG(Trusted Computing Group) が規格を策定しているセキュリティに関する機能を持つデバイスまたはチップである TPM への対応．TPM を用いたセキュリティ機能として，TrustedBoot がある．

### 4.2 GCC における実装

**Stack Protector**

ユーザ空間のスタック領域保護機能．スタック領域中にカナリア領域を作成することにより，ローカル変数のオーバフローによる値の書き換えを防ぐ．

**Built as PIE**

コンパイル時に”-fPIE –pie”オプションを指定したものは位置独立実行形式のバイナリとなり，Exec ASLR を有効にして実行する．

**Built with Fortify Source**

“-D_FORTIFY_SOURCE=2” (and -O1 or higher) でコンパイルしたものは，コンパイル時や実行時に，spritf や strcpy を，長さ制限のあるものへの置き換えや，フォーマット文字列攻撃を防止するために，フォーマット文字列のうちの%n を読み取り専用の変数にしか使用させないようにする．また，system()，write()，open() 関数の引数のチェックを行う．

**Built with RELRO**

プログラム実行時に，ローダで書き込みが必要であるが，ロード以降に書き込みが必要でないセクションは読み取り専用にする．

**Built with BIND_NOW**

ELF の遅延リンク機能を無効にし，上記の Built with RELRO と組みわせることにより，GOT(Global Offset Table) を読み取り専用にする．遅延リンク機能は，共有ライブラリのシンボル解決をロード時ではなく，実行時に行うもので，遅延リンク機能が有効である場合，共有ライブラ

リのエントリを持つ GOT は書き換え可能でなくてはならない．遅延リンク機能が有効である場合，攻撃者は GOT のエントリを書き換えることにより，任意のコードの実行が可能となる．

**Built with -fstack-clash-protection**

スタックガード領域サイズよりも大きくスタックポインタを移動させ，スタックガード領域による保護を無効にする攻撃の対策機能．GCC にてコンパイルする際に，”-fstack-clash-protection”を加えることで，スタックポインタ移動時に，1 ページ毎アクセスを発生させ，スタックガード領域を飛び越えられないようにする．

**Built with -fcf-protection**

GCC にてコンパイルする際に，”-fcf-protection”オプションを加えることで，ROP(Return-Oriented Programming) 攻撃のハードウェアでの防止機能である Intel CET(Control-flow Enforcement Technology) を利用する．

### 4.3 glibc における実装

**Password hashing**

/etc/shadow などに使用される glibc の crypt() 関数のハッシュアルゴリズムに SHA-512 を使用する．

**Heap Protector**

ユーザ空間のヒープ領域保護機能．MALLOC_CHECK_ 環境変数により，機能の機能を切り替えられ，malloc() により動的メモリ確保されたメモリ領域が free() によって解放された時に，確保されていたメモリ領域の手前 4 バイトと後続 1 バイトが不正な値で上書きされていないか，free() したポインタが，再度 free() されていないかを検査する．

**Pointer Obfuscation**

glibc の中のポインタ難読化機能．暗号化を行う PTR_MANGLE マクロと復号化を行う PTR_DEMANGLE マクロを使用し，glibc 中のポインタに難読化を行い，寿命の長い関数ポインタなどの悪用を難しくする．

### 4.4 Linux Security Module における実装

**AppArmor®**

パスベースの強制アクセス制御を提供する Linux Security Module(LSM) である．AppArmor は，プログラム毎に規則を設定し，リソースへのアクセスを制限する．規則が設定されていないプログラムに関しては，リソースへのアクセス制御を行わない，ブラックリスト型の強制アクセス制御を提供する．

**SELinux**

inode ベースの強制アクセス制御を提供する LSM である．SELinux は，ファイル，プロセス，ソケットなどのオブジェクト全てに対してラベルを付与することを強制し，付与するラベルに応じてアクセス制御を行う．ラベルに対して規則が設定されていない場合は，全てのアクセスを拒否するため，ホワイトリスト型の強制アクセス制御を行う．

**SMACK**

inode ベースの強制アクセス制御を提供する LSM であり，SELinux よりも管理が容易であることを目指したものである．SELinux と同様にオブジェクトに対してラベルを付与し，ラベルを用いてアクセス制御を行うが，SELinux におけるユーザのアクセス権限の集合であるロールや，ファイルへの強制的ないラベルの付与を省略している．

### 4.5 上記以外における実装

**No Open Ports**

Acahi(ZerocConf) で実現された機能．Ubuntu インストール後に，デフォルトで 80 番や 443 番などの特定ポートは，Listen 状態にしない．

**Automatic security updates**

GNOME® Update Manager で実現された機能．セキュリティに関するアップデートを，自動的に行う．

**Configurable Firewall**

ufw(Uncomplicated FireWall) で実現された機能．iptables のフロントエンドである ufw の提供を行う．

**Cloud PRNG seed**

Pollinate パッケージで実現された機能．擬似乱数生成時の seed を専用のサーバから取得する．

**Encrypted LVM**

Ubuntu インストーラで実現された機能．暗号化された論理ボリュームに Ubuntu をインストールする．

**File Encryption**

eCryptfs で実現された機能．暗号化されたディレクトリの作成を可能にする．

## 5. IoT 機器のセキュリティ機能の現状と今後

IoT 機器に利用される OS のセキュリティ機能は，スタックオーバフローを防止するものが主であった．また，それらの実装方法は MPU の有無で大きく異なる．MPU を用いた実装方法は，スタック領域の境界に読み取り専用領域を作成するというものである．この実装は，Linux と同様の方法であり，実装方法がある程度確立されていると言える．MPU を用いない実装方法としては，FreeRTOS のスタックポインタ位置の比較，スタック領域の境界の値の比較による実装と Zephyr の MSPLIM と PSPLIM を用いる実装がある．FreeRTOS のスタック領域の境界の値の比較は，境界の値が公開されているため，回避が可能であり，効果的な方法とは言えない．スタックポインタ位置の比較方法は，コンテキストスイッチ時に検証するのみであり，それまでの間に領域外への書き換えとスタックポインタの位置を戻すことにより回避が可能であると考えられる．Zephyr の MSPLIM と PSPLIM を用いる実装は，FreeRTOS のスタックポインタ位置の比較方法を CPU 上で常に行うもの

ⓒ 2020 Information Processing Society of Japan

であり，効果的な対策方法であると考えられる．MSPLIM と PSPLIM を用いた実装が少ない原因としては，これらのレジスタは ARMv8-M アーキテクチャから追加されたものであり，OS での実装がアーキテクチャの機能に対して追いついていないからであると考えられる．今後，OS の実装が進めば，ARMv8-M アーキテクチャを採用した，MPU を搭載していない組込みシステムであっても，スタックオーバフローからの保護機能は提供されやすいと考えられる．

Linux が実装していて組込みシステムに使用される OS が実装していないセキュリティ機能は，多く存在するが，Linux 以外にも搭載されるようなセキュリティ機能として，ヒープ領域保護，ASLR，kASLR，SecureBoot が挙げられる．ヒープ領域保護は，Linux の実装においても，特別なハードウェアや環境が必要であるものではないが，ASLR と kASLR に関しては，メモリ空間の大きさが必要になる．本論文で対象としている組込みシステムは，汎用システムに比べてメモリ空間が大きくないため，ASLR や kASLR の機能を組込みシステムで Linux など汎用システムが採用している手法で実装を行ったとしても，効果的ではない．また，ASLR，kASLR だけでなく，資源の潤沢さを用いて攻撃を困難にするようなセキュリティ機能は，資源の乏しい組込みシステムに同じ手法で実装することは意味をなさないと言える．SecureBoot は，実行する OS が正しいものか検証するセキュリティ機能であり，潤沢な資源を必要とするものではないが，実行する OS を検証するソフトウェアや検証するための信頼できる基盤が必要となる．

Linux が実装しているセキュリティ機能には，Linux 特有のものが存在する．例えば，/proc/$pid/maps protection や，Kernel Address Display Restriction などである．これらは，Linux の構造上，必要とされるセキュリティ機能であり，IoT 機器にはこれらの機能は必要がない．今後，Linux のセキュリティ機能を IoT 機器に適応できるかどうかを考えていく上で，それぞれのセキュリティ機能が，IoT 機器上で効果的であるか，また，実現可能，実現不可能，実現不要であるかどうかについても検討を行っていく必要がある．

## 6. おわりに

本論文では，IoT 機器の CPU として採用が広がっている ARM Cortex-M シリーズ上で動作する OS に必要なセキュリティ機能を明らかにするために，IoT 機器に利用される OS のセキュリティ機能の実装に利用されている ARM Cortex-M シリーズの機能を調査し，それぞれの OS のセキュリティ機能とその実装を調査することで，IoT 機器のセキュリティ機能の現状を明らかにした．そして，IoT 機器に比べて機能が豊富な汎用システムではどのようなセキュリティ機能が存在するかを明らかにするため，Linux のディストリビューションの 1 つである Ubuntu が搭載しているセキュリティ機能を調査した．今後は，Ubuntu が搭載しているセキュリティ機能を中心に，MMU を持たない機器で動作する Linux Kernel では，どのセキュリティ機能が動作するのか，また，動作しないセキュリティ機能はなぜ動作しないかを調査し，IoT 機器で実装可能，実装不可能，実装不要と分類し，IoT 機器でどのようなセキュリティ機能が有効とされ，実現可能であるかの検討を進め，実装を目指す．

### 参考文献

[1] サイバーセキュリティ研究所サイバーセキュリティ研究室：NICTER 観測レポート 2018，https://www.nict.go.jp/cyber/report/NICTER_report_2018.pdf (2019).
[2] 宮田健：IoT デバイスを狙うマルウェア「Mirai」とは何か――その正体と対策，https://techfactory.itmedia.co.jp/tf/articles/1704/13/news010.html (2017).
[3] 総務省：IoT 機器調査及び利用者への注意喚起の取組「NOTICE」の実施，https://www.soumu.go.jp/menu_news/s-news/01cyber01_02000001_00011.html (2019).
[4] Arm Limited: *ARM®v7-M Architecture Reference Manual* (2014).
[5] Arm Limited: *ARM®v8-M Architecture Reference Manual* (2017).
[6] Eclipse Foundation: IoT Developer Survey 2019 Results, https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2019.pdf (2019).
[7] FreeRTOS: FreeRTOS - Free RTOS Source Code Downloads, the official FreeRTOS zip file release download, https://www.freertos.org/a00104.html (2019).
[8] RIOT OS: Release-2019.07, https://github.com/RIOT-OS/RIOT/releases/tag/2019.07 (2019).
[9] Zephyr: Zephyr 2.0.0, https://github.com/zephyrproject-rtos/zephyr/releases/tag/zephyr-v2.0.0 (2019).
[10] Ubuntu Wiki: Security/Features, https://wiki.ubuntu.com/Security/Features (2019).

ⓒ 2020 Information Processing Society of Japan