Computer Security Symposium 2019
21 - 24 October 2019

TrustZoneを利用した安全なメモリ操作によるプロセス状態確認手法

青木 和也[1,a)]　掛井 将平[1]　瀧本 栄二[2]　毛利 公一[2]　齋藤 彰一[1]

概要：マルウェアの高度化によりあらゆる異常動作を検知することは難しくなっている．アンチウイルスソフトを回避するためにファイルを生成しない，ファイルレス型のマルウェアも存在する．実行中のプログラムの動作を正確に解析するにはメモリダンプでメモリ上に展開されたプログラムを取得し解析する必要がある．しかしメモリダンプを防ぐ技術も存在するので安全にメモリダンプを取得する方法は確立していない． 本論文では ARM のセキュリティ拡張機能である TrustZone を利用する．TrustZone は計算機資源を Normal World と Secure World と呼ばれる 2 つの領域にハードウェア的に分割する．Secure World は Normal World より高い権限を持つので Normal World で動作するプロセスは Secure World での処理を妨げることは難しい．Secure World から Normal World のプロセスメモリを解析するするためのプロトタイプを実装し，機能の一例としてプロセスの正常動作を保証できることを確認した．

キーワード：Trustzone，プロセスメモリ，検証， 解析，OP-TEE

An Introspection Method of Process Memory with Trusted Opetation

Kazuya Aoki[1,a)]　Shohei Kakei[1]　Eiji Takimoto[2]　Koichi Mouri[2]　Shoichi Saito[1]

Abstract: The advancement of malware development has raised difficulty of anomaly detection in various ways. Some malware avoids detection of anti-virus software called fileless malware. Hence, it is essential to obtain and analyze programs loaded on memory for accurate introspection of running programs. However, no method for generating memory dumps safely has been established. In this paper, we leverage TrustZone, a security extension of ARM, that allows software executed to be split in two environments: normal world and secure world. Since the secure world has higher level of privilege than the normal world, processes in the normal world cannot interfere with the secure world. We implemented a method in the secure world for introspection of process memory loaded in the normal world. We evaluated that our method is works properly, and the results show that it can verify whether a process is running normally.

Keywords: TrustZone, Process Memory, Introspection, Memory Forensic, OP-TEE

1. はじめに

スマートフォンや IoT 機器の普及によって攻撃の対象が増加し，種類も多様化している．マルウェアや不正アクセスによる攻撃とアンチウイルスソフトによる防衛は双方ともに高度化している．マルウェアには解析環境を検知して動作を変えるものやコードの難読化，特定の条件下でのみ悪意のあるコードを実行するといった特性を持つものが知られている [1][2]．攻撃は一つの脆弱性を攻略することができれば成功することから，全ての穴を塞がなければならないセキュリティには限界がある．よって，サーバに侵入されたり端末が乗っ取られたとしても攻撃の目的である秘

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 立命館大学
Ritsumeikan University
[a)] k.aoki.709@nitech.jp

密情報だけは取られたり破壊されないことが重要である．

モバイル機器やIoT機器の多くにはARM社製のSystem on Chip（SoC）が使われている．これらには不正アクセスから秘密情報や認証処理，暗号処理を保護するための機構としてARMプロセッサで利用可能なTrustZoneを提供している．TrustZoneはTrusted Execution Environment(TEE)を実現するためのARMアーキテクチャのセキュリティ拡張機能である．また，TEEとはソフトウェアだけでなくハードウェアによるサポートにより，アプリケーションの安全な実行環境を実現するための技術仕様である．GlobalPlatform[3]というICカードに関連する業界標準化団体により，TEEに提供される各種Application Programing Interface(API)の仕様策定が行われている．

TrustZoneはメモリをNormal WorldとSecure Worldと呼ばれる領域に分離し，秘密情報をSecure Worldに隔離することで安全な実行環境を提供する．それぞれの領域では別々のOSが動作しており，Normal Worldでは一般的なプロセスと汎用OS(Linux,Android)が動作する．一方，Secure Worldでは高いセキュリティが求められるプロセスや，汎用OSと比較して小さなセキュアOSと呼ばれるOSが動作する．例えば，暗号処理について述べると，Normal WorldからSecure Worldへのリソースアクセスは特定のAPIを通してのみ行われ，暗号処理の結果のみを返す．逆にSecure WorldからNormal Worldへのアクセスは制限されない．近年の汎用OSは豊富な機能やモダンなインターフェースを提供するために膨大なサイズになっており，様々なプロセスが動作することから脆弱性が存在する可能性が高い．脆弱性によってNormal Worldに存在する汎用OSが乗っ取られたとしてもSecure Worldへのアクセスは限られているため秘密鍵や暗号処理の途中結果といった秘密情報は保護できる．つまり，TEE上で実行されているプロセスとデータにはTrustZoneの実装に脆弱性が無い限り，汎用OSから不正アクセスすることは出来ない．

本論文では主にプロセスメモリを対象として，モバイルデバイスやIoT機器のNormal Worldで動作する不審な動作や痕跡をSecure Worldから監視するためにTrustZoneを活用する．監視対象のOSに対して別のOSから検査を行う方法の一つにVirtual Machine Introspection(VMI)[4]がある．これを利用することでVMのモニタリングを行う管理者はホストOSから実行中であるゲストOSのメモリ内データやカーネルのデータ構造を参照できる．そのため，ゲストOSにアンチマルウェアのためのアプリケーションを直接インストールする必要がなくなる．さらに，VMIによりハイパーバイザを通して仮想マシンの状態の取得や制御を行うこと(introspection)ができる．TrustZoneが利用できる環境では，セキュアOS内にVMIツールに相当する機能を追加することで同様のことを実現できる．このように，セキュアOSからプロセス情報の取得，特定のアドレスのデータに対するintrospectionを行う手法を提案する．Normal WorldからはSecure Worldの処理に干渉できないこと，Secure Worldの処理はCPUのコアを占有することでマルウェアは解析されていることに気づかない特性により，提案手法の安全性を保証する．また提案手法はARM社製のSoCを搭載した機器のみで利用でき，多くのマルウェア解析のように解析用のマシンを用意する必要はないという利点がある．

本論文の構成は次の通りである．まず，第2章ではTrustZoneのメモリ分割機能を利用した先行研究について述べる．次に第3章でTEEのオープンソース実装であるOP-TEEの構成について述べる．第4章では汎用OSで動作するプロセスのメモリをintrospectionするための提案手法の概要とそのユースケース，アーキテクチャについて述べたあと，第5章で提案手法の実装について述べる．そのあと，第6章で提案手法の安全性について議論を行い，最後に第7章で本論文のまとめを述べる．

2. 関連研究

本章ではTrustZoneを活用してNormal Worldのセキュリティを向上させるためのメモリ分割に関係した研究について述べる．TrustZoneの本来の活用例は秘密鍵といった秘密情報をSecure Worldに配置して保護する，認証処理をSecure World内で行うことで安全を保証することだが，そのほかにも応用的に様々な活用方法が提案されている．まず，TrustZoneと比較されるIntel Software Guard eXtensions(SGX)[5]の利点を取り込んだ研究について述べる．次にCPUからみてより権限の高い領域を用意する考えではなく，イントラレベルで権限分離を行うことで各領域で保護領域を作ることを実現した研究について述べる．最後に，Secure WorldからNormal Worldのリソースへ一方的にアクセスできるという特性を利用してアプリケーションがマルウェアかどうかをAPI呼び出し情報とシステムコールを入力として機械学習により判別した研究とNormal Worldのメモリを Secure Worldから取得するためのVMIライブラリの研究について述べる．

2.1 Enclaveの概念をTEE環境で実現

Trusted Application(TA)と呼ばれる，Secure Worldで動作するアプリケーションに脆弱性が存在するとセキュアOS全体の安全性が侵害される．それによって，TAには厳しい制限が課せられており，OS開発を行うデバイスベンダーとアプリケーション開発者には相互の信頼が必要になる．SANCTUARY[6]ではデバイスベンダーの信頼が無くてもTAの保護機構の特性を活かすためにIntel SGXのEnclaveという概念を導入する．Intel SGXはメモリ内に存在する特定アプリケーションのコードとデータを隔離するためのハードウェア支援によるメモリ暗号化機能を備えて

いる．ユーザレベルのコードを Enclave と呼ばれるプライベート領域のメモリに割り当てることで，細かい粒度で制御と保護を実現する．TrustZone は Normal World と Secure World を切り替えるためのコンテキストスイッチが必要になるが，SGX は Normal World 内のメモリの一部を暗号化するため，メモリ切り替えのためのコンテキストスイッチによるオーバーヘッドを必要としない．SANCTUARY では ARM アーキテクチャで Enclave を実現するために，CPU バスを解釈して処理を行う CPU のコアを分けるモジュール (TrustZone Address Space Controller) を改変し，Enclave を作成するためのメモリ領域と専用コアを用意する．Enclave 内にはその外からアクセスすることはできないので TrustZone における TA の安全性と同じ原理で安全に Normal World 内におけるプロセス実行を実現している．

2.2 イントラレベル権限分離

既存のものより権限の高いモジュールを追加し，そこでモニタリングツールを動作させることでセキュリティを高める考えとは別に，イントラレベルで権限分離を行うアプローチがある．イントラレベルの権限分離とは CPU からみると同じ権限で実行されるメモリ領域だが，プロセスからはアクセス出来る範囲が制限されていることである．Hilps[7] は 64bit の ARM アーキテクチャに登場した TxSZ というハードウェア機能を活用することで，初めてイントラレベルの完全な権限分離を実現した．汎用 OS やセキュア OS だけでなく，ハイパーバイザーも含めてそれぞれのメモリ領域をインナードメインとアウタードメインの二つに分離する．アウタードメインからインナードメインへのアクセスは制限され，その逆は制限されない環境を実現する．セキュリティが求められるモニタリングツールをインナードメイン内で動作させることでアウタードメインをモニタリングできる．またセキュア OS 内も 2 つのドメインに分割することで，TA の脆弱性がアウタードメインに存在してもインナードメインには干渉することが難しくなり全体のセキュリティ向上が期待できる．しかし，現在の Hilps ではページフォルト等の意図しないエラーによってセキュリティツールがクラッシュする可能性がある．

2.3 マルウェアの動的解析の研究

Android アプリケーションの不審な振る舞いを検知するための様々な研究 [8][9] が存在するが，難読化やアンチウイルスソフトの無効化を備えたマルウェアが多く見られる．ハイパーバイザや Virtual Machine によって保護機構を分離して実行するアプローチはオーバーヘッドを考慮すると現実的ではない．T2Droid[10] では TrustZone を利用して Secure World でマルウェアを判別する．Android API 呼び出しとカーネルのシステムコールを動的解析によって取得したあと，機械学習によりアプリケーションが不審かどうかを判別する．この研究ではコンテキストスイッチによるオーバーヘッドの増加を抑制するために Android API とシステムコールを追跡する処理は Normal World にある．このため，この追跡をマルウェアに検知される可能性がある．

2.4 Introspection

ITZ Library[11] では TrustZone を利用して Secure World から Normal World のメモリ状態を解析するためのライブラリを提供する．Normal World 上に未知のプログラムが動作していたり，汎用 OS 自体が信頼できない環境でも安全にモニタリングできることを目指している．このシステムでは，LibVMI[12] という VMI のための Python でバインディングされた C ライブラリで提供される機能を TrustZone の環境で利用可能にしている．Normal World の仮想アドレスや物理アドレスを指定すると Secure World からそのメモリの読み書き，値の表示を行う．また，ITZ Library を使った introspection のツールにはカーネルの正当性チェックやハッシュ値の計算をするものがある．この研究では一部のメモリ操作を行うことできるが，異常検知という点では利用範囲がカーネルのみに限られる．本提案ではこれらの機能と合わせてプロセス状態の確認を行う．

3. OP-TEE

本章では，提案手法の実装に用いた OP-TEE[13] という TEE 実装のデザインと汎用 OS，セキュア OS 間における通信について述べる．

3.1 デザイン

OP-TEE とは GlobalPlatform の API 仕様に基づき，TrustZone を用いてコードとデータの完全性と機密性を確保するための独立環境 (TEE) を提供するオープンソース実装である．OP-TEE は Normal World のユーザ空間に存在する TEE Client API，Linux カーネル内の TEE デバイスドライバ，セキュア OS である Optee OS の 3 つで構成される．**図 1** に構成を示す．OP-TEE は GlobalPlatform の定義する API を 2 種類利用していて，それぞれ TEE Client API と TEE Internal API と呼ばれる．Secure World で動作するソフトウェアのことを Trusted Application(TA) という．TA では認証処理や暗号処理といったセキュリティが求められる処理を行う．Optee OS は常には動作しておらず，クライアントアプリケーション (CA) からの依頼によって動作を開始しない限り CPU リソースを消費することはない．Optee OS は特定のハードウェアシグナルに対してイベントハンドラを登録しており，SMC（セキュアモニタコール）命令や IRQ，FIQ，外部アボートによるイベントが発火すると登録された動作を行う．World の切り替えを行うためのレジスタの書き換えはセキュアモニタ

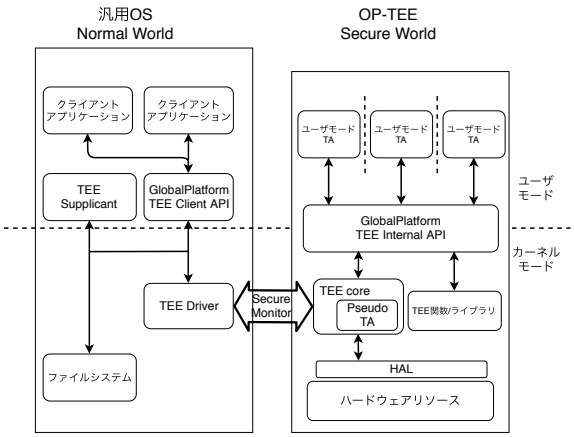

図 1 OP-TEE の構成
Fig. 1 Components of OP-TEE.

モードで行われる．TA にはユーザモードで動作するユーザモード TA とカーネルモードで動作する Pseudo TA が存在する．ユーザモード TA は TEE Internal API や TEE のライブラリを利用して処理を行う．Pseudo TA はセキュア OS と同等の権限を持っている．これは CA から直接呼び出しを行わないようにし，ユーザモード TA から呼び出すように TA 開発者が設計することで，クリティカルな処理を CA に頼らないようにすることができる．

3.2 GlobalPlatform API による World 間の通信

TEE Client API は TEE と通信するために汎用 OS でどのように操作を行うべきかを定義している．Normal World から Secure World にデータを送信したり，その結果を受け取るときに利用することができる．32bit の整数を 2 つかまたは，unsigned char 型のポインタを Secure World に送信する 1 つのデータとして使用することができる．一度に送ることができるデータの数は実装に依存する．通信する TA の指定には TA 開発者が TA 毎に定義した UUID を使用する．また，TA で行う処理の指定には関数単位の処理を表す commandID を指定する．

4. 提案手法

本論文では TrustZone を利用して，ARM デバイスで動作する汎用 OS のプロセスメモリ監視手法を提案する．2.4 節の研究では特定のアドレスのメモリ値に対して読み書きや表示を行っているが，それだけではマルウェアの検知やプロセスの introspection という目的に対しては実用的ではない．また，限られた開発ボードでしか動作しないため，利用範囲も制限される．本章では提案手法の概要について述べたあと，その活用場面と全体の構成について述べる．

4.1 概要

既存手法では Secure World から Normal World のメモリに対して読み書き等，単純な操作しか行うことができない．本提案ではプロセスメモリに焦点を当てて，プロセス実行の監視を行うことでプロセスメモリの正当性の保証を実現する．Normal World のプロセスメモリの取得，確認，操作を Secure World から行うことでアンチウイルスソフトの無効化機能を備えたマルウェアにも対応できる．安全にプロセスメモリを取得することが可能なので，攻撃への対応も含めた異常検知のための監視という視点で利用できる．また，安全に Secure World から Normal World のメモリ操作を行えるという特徴を生かすことで，信頼できない汎用 OS や他に不審なプロセスが動作している環境においても，プロセス状態の正当性を保証することを可能にする．

4.2 ユースケース

ユースケースにはプログラマが自身の管理するプログラム内で自己プロセスのメモリを確認できることと，未知のプロセスに対してメモリダンプを行い，それの詳細な解析を行うことが挙げられる．前者は Bring Your Own Device(BYOD) を採用している会社で社員が自身のデバイスを持ち込み，業務を行う状況を想定する．Normal World 内でどんなプロセスが動作しているか分からない環境でも，秘密情報へのアクセスといったクリティカルな操作を安全に行うために利用する．**図 2** に状況を示す．企業のデータベースやクラウドにアクセスするための企業が作成した CA に対して，Secure World にある提案手法の TA が introspection を行う．TA が正常な CA だと判断した場合のみ，Secure World 内の秘密鍵で作成された証明書を CA に返すことで正当性を確認した CA のみが秘密情報にアクセスする．また，異常だと判断できる場合にデバイスを自動的にロックするなどのセキュリティ向上対策を行うこともできる．後者はメモリダンプを複数回行うことでプロセスの状態遷移を確認することと，ダンプファイルに詳細な解析を実施してマルウェアの痕跡を探すことができる．ダンプファイルにはダンプ解析によく使われている volatility[14] の機能を参考にして実装したダンプ解析を行う．

4.3 アーキテクチャ

提案手法の全体図を**図 3** に示す．本論文では introspection を行う対象のプロセスをターゲットアプリケーションと呼ぶ．まず，クライアントアプリケーションがターゲットアプリケーションの PID を元に proc ファイルシステムからプロセスのメモリマップを取得する．それを元に introspection を行う開始アドレスである仮想アドレスとメモリサイズを決定する．introspection を行う範囲は目的によって異なり，プロセスメモリのコード部，静的データや動的に取得するメモリ，プロセスメモリ全体がある．プロセスメモリはユーザ空間とカーネル空間に分けられる．仮想アドレスがユーザ空間なら proc ファイルシステムを利用

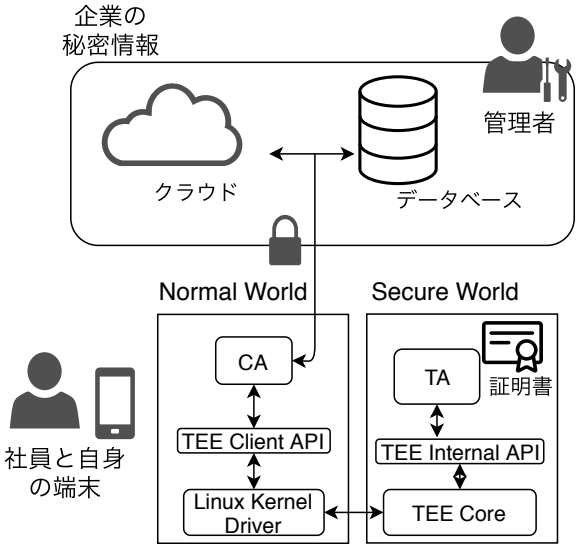

図 2 BYOD から企業の秘密情報にアクセスするユースケース
Fig. 2 A use case for accessing confidential company information from BYOD.

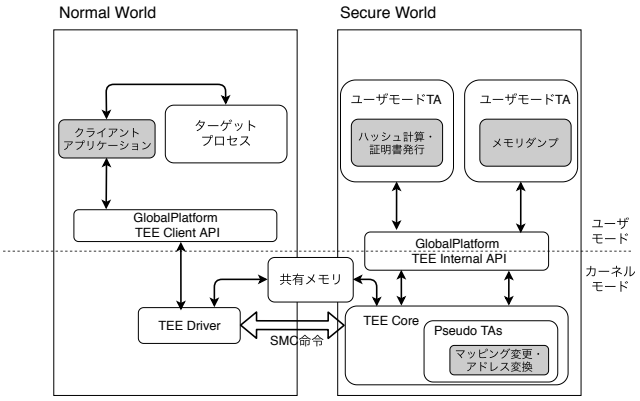

図 3 提案手法
Fig. 3 Proposal.

表 1 実装環境
Table 1 An implimentation environment.

Board	Raspberry Pi 3 ModelB
OS	Linux/OP-TEE(3.4.0)
CPU	Broadcom BCM2837
Memory	1GB

することで仮想アドレスから物理アドレスへの変換可能である．ここではターゲットアプリケーションは実行状態であり，物理メモリは十分にあると仮定して，ページングの影響は受けないものとする．また仮想アドレスがカーネル空間の場合はカーネル関数である `virt_to_phys()` をカーネルモジュールから呼び出すことで対応する．最後に物理アドレスを TEE Client API のパラメータとしてユーザモード TA に送信し，Pseudo TA でセキュア OS のマッピングに追加することでセキュア OS の仮想アドレスに変換できる．これらの操作によって Secure World から Normal World のメモリに対してアクセス可能になる．

一方で，OP-TEE では Normal World において仮想メモリを共有メモリとして指定することで，Normal World と Secure World の双方からアクセスできるメモリ領域を作成することが可能である．この方法の利点は物理アドレスへの変換が不要でメモリスワップに対応できることと，TEE Client API によって CA と TA で効率よくデータ共有ができることである．しかし，ターゲットアプリケーション内で共有メモリに関する API を利用しなければならず，CA が作成することはできない．物理アドレスをパラメータとして渡す方法ではこの制約は受けない．

本提案の提供するメモリ操作に関する機能について述べる．ユーザモード TA ではメモリの値の読み書き，値の表示，ハッシュ値の計算，メモリダンプの取得を実現する．ハッシュ値を求めることで秘密情報に変更がないか，フローの条件が正常かどうかが確認できる．Pseudo TA ではセキュア OS のメモリマップに関係する操作とセキュア OS における物理アドレス，仮想アドレスの変換を行う．

5. 実装

OP-TEE には ARM 社製の Cortex-A 系プロセッサを搭載したボード向けのビルド用のスクリプトが用意されているので利用可能な範囲が広い．Android 端末や多くの IoT 向け開発ボードに対応している．今回はこれを改変することで実装コストを減らすとともに広く利用可能にすることを実現した．実装環境を**表 1** に示す．

本章では提案手法を実現するために OP-TEE における World 間のアドレス変換について述べたあと，提案手法の各要素における処理と introspection の流れについて述べる．

5.1 OP-TEE における World 間のアドレス変換手法

本節ではシステム設定，メモリマップについて述べたあと，アドレス変換手法について述べる．

5.1.1 システム設定

TEE 環境ではセキュアブートと呼ばれる，起動時のカーネルやファームウェアの正当性を担保する技術がある．物理的に改ざんが難しいハードウェアを信頼の基点として次に起動するソフトウェアの正当性を確認するものである．

OP-TEE ではセキュアブートによって非 TEE 環境とは異なる OS の起動動作を行う．起動の流れについて述べる．まず，ブートローダが Optee OS を起動したあと，Optee OS は自身の初期化を行う．その次に Normal World にコンテキストスイッチを行い，汎用 OS が起動される．よって，OP-TEE 環境では Optee OS がシステム設定から物理メモリを確保したあとに汎用 OS が物理メモリを確保する

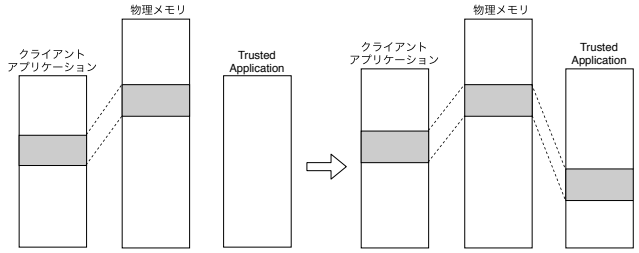

図 4　メモリマップの変化
Fig. 4　Memory map changes.

ことになる．すでに Optee OS が確保した物理メモリは汎用 OS から見ると予約済みとなり，同じ物理メモリを確保することはできない．よって必要に応じて共有メモリのサイズを変更し，introspection を行うのに十分な領域を確保する必要がある．また Optee OS の設定で 1 つの TA 確保するメモリ領域が定められているため，これも変更する必要がある．

5.1.2　メモリマップ

汎用 OS のメモリ管理はセキュア OS にとって未知であるため，汎用 OS で扱う仮想アドレスでは TA からアクセスすることはできない．汎用 OS の管理するメモリにセキュア OS の TA からアクセスするためには仮想アドレスの変換が必要になる．ここでは仮想アドレスを物理アドレスに対応づけることをマッピング，それぞれのアドレスを対応づけの関係を VA-PA マップという．VA-PA マップの変更にはシステムにより起動時に決定する方法と TEE Client API やカーネル関数により動的に変更する方法がある．事前に全ての物理メモリを Secure World にマッピングすることは実装上困難であるので，ターゲットアプリケーションに関係するプロセスメモリに対してのみ動的に指定する．

VA-PA マップの変更について**図 4** に示す．このようにターゲットアプリケーションの仮想アドレスに対応する物理アドレスを共有メモリとして Secure World の VA-PA マップに追加する．これによって間接的に Normal World の管理するメモリを Secure World から操作可能になる．

5.1.3　アドレス変換

Normal World と Secure World の仮想アドレスは同じ物理アドレスに対して一般に異なり，Secure World から Normal World のメモリ操作を行うには目的の物理アドレスに対応した Secure World の仮想アドレスを取得するためのアドレス変換が必要になる．アドレス変換には (1) 物理アドレスを Secure World の仮想アドレスに変換，(2)Normal World の仮想アドレスを Secure World の仮想アドレスに変換，(3) 共有メモリのアドレスを Secure World の仮想アドレスに変換の 3 通りがある．

1 つ目は Normal World から送信された物理アドレスの値を Secure World において VA-PA マップに追加する．OP-TEE のカーネル関数である `tee_core_mmu_add_mapping()` を利用すると，対象の物理アドレスを共有メモリとして指定できる．その次に `phys_to_virt()` で物理アドレスとメモリタイプを指定することで Secure World の仮想アドレスに変換できる．2 つ目は Normal World の仮想アドレスを物理アドレスに変換したあとで，それを Secure World の仮想アドレスに変換する．まず，proc ファイルシステムの/proc/PID/pagemap を参照することで Normal World にあるプロセスのユーザ空間に属する仮想アドレスを物理アドレスに変換できる．また Linux のカーネル関数を使うことでカーネル空間の仮想アドレスは物理アドレスに変換できる．物理アドレスから Secure World の仮想アドレスへの変換は，前述の 1 つ目の変換と同様にして変換する．3 つ目は Normal World で共有メモリを設定する方法である．この方法は CA とターゲットアプリケーションが同一である必要がある．共有メモリの開始アドレスとサイズを引数として TEE Client API の `TEEC_SharedMemoryRegister()` により任意のアドレスを動的に共有メモリとして指定できる．この方法では物理アドレスへの変換を挟む必要がない．流れとしては，まず Normal World で共有メモリを作成し，その先頭アドレスとサイズを Secure World へ送る．Secure World では `TEE_MemMove()` によって当該共有メモリの Secure World における仮想アドレスを取得できる．以上 3 つの方法のいずれかを行うことで，TA はターゲットアプリケーションのメモリ操作を行うことが可能になる．

5.2　introspection 手法

本節では図 3 におけるクライアントアプリケーション (CA) とユーザモード TA，Pseudo TA の役割や処理の流れについて述べる．

CA は提案手法の起点となるプロセスであり，introspection はこの CA の要求に基づいて実行される．CA は TA を呼び出す前に，/proc/PID/maps より introspection を行う仮想アドレスの範囲を決定する．これに対して 5.1.3 節の (1) か (2) の方法でアドレス変換を行う．ここで，仮想アドレスでは連続でも物理アドレスでは連続とは限らないことに注意して，アドレス変換をページ単位で行う．物理アドレスをセキュア OS の VA-PA マップに追加するためには Optee OS のカーネル関数を使用するため，Pseudo TA を用いる．Pseudo TA はユーザモード TA から呼び出されて Secure World の仮想アドレスを返す．ユーザモード TA では CA の要求に基づき，メモリの読み書き，表示，ハッシュの計算や証明証の発行，メモリダンプの実行を行いその結果を CA に返す．ハッシュ値を計算するライブラリは，利用した OP-TEE のバージョンには存在しないため，SHA256 アルゴリズムと BASE64 エンコードを実装して追加した．なお，メモリダンプの実行はダンプファイル保護するために，Secure World にある鍵で暗号化した上で

Normal World のファイルシステムに置くための技術である Secure Storage を利用し，平文のままファイルシステムに置かないようにした．

6. 評価

本章では提案手法の安全性について議論する．また，メモリに対する攻撃例を紹介したあと，それが提案手法により検知可能であることを示す．

6.1 安全性評価

本節ではマルウェアによって提案手法が無効化される可能性について議論する．

6.1.1 Normal World にある CA の偽装

一般に，TA は複数の CA から呼び出されることが前提の仕様になっているが，セキュリティツールとして TA を利用する場合はこの前提にはリスクがある．TA が呼び出される前に CA のコードセクションのハッシュ値を計算し，Secure World に置かれたハッシュテーブルを参照することでホワイトリスト形式の認証を行う研究 [15] がある．これを利用することで CA と TA を一対一の関係にすることが可能である．これによって CA の偽装を防ぐ．

6.1.2 TA への侵害

Secure World に Normal World から侵害することはできないことから，Normal World のマルウェアが TA の動作に干渉することはできない．そのため，保存中のバイナリファイルの安全性を検討する．OP-TEE では，TA のバイナリファイルの保存場所によって次の 3 種類が利用できる．Normal World のファイルシステムに置かれた TA，Secure World からしか復号できないよう Secure Storage を利用して暗号化した TA，Secure OS のコアにあるスペシャルデータセクションにリンクされている TA である．これらの内で，Secure Storage に TA を置く方法を採用することで暗号化された TA の実行ファイルに対して管理者に気づかれることなく細工をすることは困難になる．

6.1.3 introspection の無効化

セキュア OS で処理が行われているときには CPU のコアが占有されることと，Secure Configuration Register の NS bit と呼ばれるモニターモードでしか変更不可能なレジスタによってメモリがどちらの World のものかを判別することが TrustZone の安全性の保障になる．よって，TEE の実装に脆弱性がない限り Normal World では root 権限を持っていても Secure World にある TA からの introspection に対して干渉できない．

6.2 Process Hollowing

本節では具体的なメモリに対する攻撃である Process Hollowing について説明したあと，その攻撃を提案手法で検知可能であることを確認する．

6.2.1 実験内容

Process Hollowing とはマルウェアが正規のプロセスを起動したのち，メモリ上のコードを悪性のコードに置き換えることでアンチウイルスソフトに検知されないことを目的とした攻撃手法である．提案手法の評価として，Process Hollowing への耐性を確認する．ターゲットアプリケーションとして “Hello, World!” の出力を繰り返す自作プログラムを用いる．このプログラムに対して ptrace システムコールを使用し，プログラムの一部である出力文字列をメモリ上で変更する攻撃を行う．ptrace システムコールの操作はカーネル内で行われるため，バイナリファイルのコード部分である.text セクションや読み取り専用の領域である.rodata セクションを書き換えることができる．

特定のアドレスを共有メモリに指定し，TA を呼び出しによってそのハッシュ値を受け取るコードを**図 5** に示す．このコードでは入力と出力の用途で使用する 2 つの共有メモリを登録したあと，`TEEC_InvokeCommand()` で TA 呼び出しを行う．この呼び出しによって，メモリの上書きによるハッシュ値の変更がないかを確認できる．また，共有メモリから値を読み取り，そのハッシュ値を返す TA のコードを**図 6** に示す．ここでは共有メモリのサイズ分，malloc でメモリを確保して，共有メモリのデータを確保したメモリにコピーしたあと，SHA256 と BASE64 でエンコードを行いその値を出力用に確保した共有メモリに書き込んでいる．return 命令が実行されると図 5 の 11 行目に処理が戻る．

実験の手順について述べる．攻撃が成功する例と提案手法を利用する例の 2 つを行う．まずターゲットアプリケーションを起動したあと，ps コマンド，objdump コマンドを用いて PID と出力文字列が格納されている変数のアドレスを調べる．それを元に攻撃用プログラムで文字列の書き換えを行う．次に文字列が出力される毎に図 5 のようにハッシュ値を計算し，その前に計算したハッシュ値と異なっていたら “Detect: hash is different” と出力して終了するターゲットアプリケーションを作成したあと，同様の攻撃を行う．

6.2.2 結果

攻撃によって出力文字列が変更される様子を**図 7** に示す．攻撃者は.rodata を書き換えて任意の文字列を書き込むことができる．ここでは攻撃により “Hello, World!” が “hacked” に書き換えられたが，ターゲットアプリケーションの実行は継続される．次に提案手法によりメモリ書き換えを検知してプロセスを終了するように変更したターゲットアプリケーションに対して同様の攻撃を実行した様子を**図 8** に示す．このように，攻撃によるメモリの変更によってハッシュ値が変更されたことを検知可能であることが分かる．

```
...
shm_in.buffer = target_addr;
shm_in.size = target_addr_len;
shm_out.buffer = hash;
// register shared memory
TEEC_RegisterSharedMemory(&ctx, &shm_in);
TEEC_RegisterSharedMemory(&ctx, &shm_out);
// invoke ta to get hash
TEEC_InvokeCommand(&sess,TA_CMD_HASH,&op,&err);
// output hash
printf("s\n", hash);
...
```

図 5　共有メモリを設定し，TA を呼び出す CA のコード

Fig. 5　CA code that allocates a specific address as shared memory and calls TA.

```
...
// shared memory size
sm_len = params[0].memref.size;
// allocate memory with mem_len
mem = TEE_Malloc(sm_len,TEE_MALLOC_FILL_ZERO);
// copy data of shared memory to mem
TEE_MemMove(mem, params[0].memref.buffer,sm_len);
// encode memory value with sha256 and base64
sha256encode(hash,mem,sm_len);
base64_enc(hash,SHA256_BLOCK_SIZE,b64,&b64_len);
// translate address
TEE_MemMove(params[1].memref.buffer,b64,b64_len);
return TEE_SUCCESS;
...
```

図 6　共有メモリから値をからハッシュ値を計算する TA のコード

Fig. 6　TA code that reads a value from shared memory and returns its hash value.

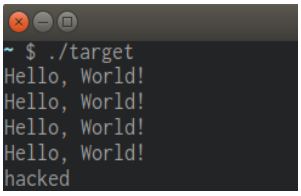

図 7　攻撃を受けた様子

Fig. 7　A state of being attacked.

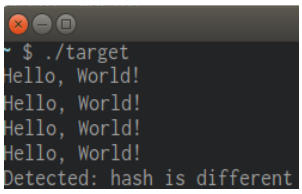

図 8　攻撃を検知する様子

Fig. 8　Detecting an attack.

7. おわりに

本論文ではプロセスメモリに着目した introspection 手法について提案し，実装を行なった．この実装はオープンソースとして公開する予定である．評価では提案手法の安全性の確認と実際のメモリ書き換えに対しての実例を示した．今後の課題としてプロセスメモリに対して操作を行う関数の充実と効率化があげられる．

参考文献

[1] Medina, R. P., Neundorfer, E. B., Chouchane, R. and Perez, A.: PRAST: Using Logic Bombs to Exploit the Android Permission Model and a Module Based Solution, *2018 13th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 1–8 (online), DOI: 10.1109/MALWARE.2018.8659369 (2018).

[2] Zhang, V.: Hacking Team RCSAndroid Spying Tool Listens to Calls; Roots Devices to Get In, `https://blog.trendmicro.com/trendlabs-security-intelligence/hacking-team-rcsandroid-spying-tool-listens-to-calls-roots-devices-to-get-in/` (Accessed on 08/13/2019).

[3] GlobalPlatform Inc.: GlobalPlatform Homepage, `https://globalplatform.org/` (Accessed on 08/16/2019).

[4] Hebbal, Y., Laniepce, S. and Menaud, J.-M.: Virtual machine introspection: Techniques and applications, *2015 10th International Conference on Availability, Reliability and Security*, IEEE, pp. 676–685 (2015).

[5] Intel: Intel® Software Guard Extensions — Intel® Software, `https://software.intel.com/en-us/sgx` (Accessed on 08/13/2019).

[6] ferdinand brasser, david gens, patrick jauernig, ahmadreza sadeghi and emmanuel stapf: sanctuary: arming trustzone with user-space enclaves, *ndss* (2019).

[7] kwon, d., yi, h., cho, y. and paek, y.: safe and efficient implementation of a security system on arm using intra-level privilege separation, *acm trans. priv. secur.*, Vol. 22, No. 2, pp. 10:1–10:30 (online), DOI: 10.1145/3309698 (2019).

[8] Felt, A. P., Chin, E., Hanna, S., Song, D. and Wagner, D.: Android Permissions Demystified, *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, New York, NY, USA, ACM, pp. 627–638 (online), DOI: 10.1145/2046707.2046779 (2011).

[9] Su, D., Wang, W., Wang, X. and Liu, J.: Anomadroid: Profiling Android Applications' Behaviors for Identifying Unknown Malapps, *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 691–698 (online), DOI: 10.1109/TrustCom.2016.0127 (2016).

[10] s. d. yalew, g. q. maguire, s. haridi and m. correia: T2droid: A Trustzone-based Dynamic Analyser for Android Applications, *2017 ieee trustcom/bigdatase/icess*, pp. 240–247 (online), DOI: 10.1109/trustcom/bigdatase/icess.2017.243 (2017).

[11] m. guerra, b. taubmann, h. p. reiser, s. yalew and m. correia: introspection for arm trustzone with the itz library, *2018 ieee international conference on software quality, reliability and security (qrs)*, pp. 123–134 (online), DOI: 10.1109/qrs.2018.00026 (2018).

[12] LibVMI Project: libvmi.com, `http://libvmi.com/` (Accessed on 08/08/2019).

[13] STMicroelectronics and Linaro Security Working Group: OP-TEE, `https://github.com/OP-TEE` (Accessed on 08/08/2019).

[14] The Volatility Foundation: The Volatility Foundation Open Source Memory Forensics, `https://www.volatilityfoundation.org/` (Accessed on 08/15/2019).

[15] Zhao, B., Xiao, Y., Huang, Y. and Cui, X.: A private user data protection mechanism in trustzone architecture based on identity authentication, *Tsinghua Science and Technology*, Vol. 22, No. 2, pp. 218–225 (online), DOI: 10.23919/TST.2017.7889643 (2017).