Arm CCAのRealm VMを活用した隔離環境でのシステム監視

高巣 雄大[1]　光来 健一[1]

概要：インターネットに接続されるシステムへの攻撃は年々，巧妙になっており，最近は機械学習を用いた高度な侵入検知システム（IDS）が用いられるようになっている．しかし，IDSを監視対象システム内で動作させると侵入者による攻撃を受ける恐れがある．そこで，隔離実行環境（TEE）を用いてIDSを安全に実行する手法が提案されている．しかし，多くの手法ではTEE内で高度なIDSを実行するのが難しく，情報を取得するためのエージェントを動作させる必要があることも多い．本稿では，Arm CCAで導入されたRealmワールドで動作するRealm仮想マシン（VM）を用いて高度なIDSを実行し，エージェントを用いずにノーマルワールドで動作するシステムを安全に監視するCCAmonitorを提案する．CCAmonitorでは，IDSがRealm VMの下で動作するRealm Management Monitor（RMM）を呼び出すことで，ノーマルワールドのメモリデータを取得してOSデータを監視する．CCAmonitorをRMMとRealm VM内で動作するLinuxに実装し，procファイルシステムによって提供されるシステム情報を取得するIDSを用いて有効性を確かめる実験を行った．

1. はじめに

近年，クラウドやIoT，エッジデバイスなど，様々なシステムがインターネットに接続されるようになっている．その結果，インターネットからシステムへの攻撃も増加している．攻撃者はソフトウェアの脆弱性や設定ミスを利用してシステムに侵入し，機密情報を盗んだり，サービスを妨害したりする可能性がある．従来型の侵入検知システム（IDS）では年々，巧妙になる攻撃を完全に防ぐことは難しいため，最近は機械学習を用いた高度なIDSが用いられるようになっている．しかし，監視対象システム内でIDSを動作させる場合，システムに侵入した攻撃者によってIDS自体が攻撃され，無力化されてしまう可能性がある．

そこで，プロセッサが提供する隔離実行環境（TEE）を用いてIDSを安全に実行し，監視対象システムのメモリ上のOSデータを監視する手法が提案されている．例えば，Intel SGXやRISC-V Keystoneなどのエンクレイヴ内でIDSを実行してホストを監視したり，Arm TrustZoneのセキュアワールドでで IDSを実行してノーマルワールドを監視したりすることができる．しかし，エンクレイヴ内では一つの小さなアプリケーションだけを動かすことが想定されているため，高度で複雑なIDSを動かすのには向いていない．セキュアワールドでは複数のプロセスを動かすことができるが，IDSが攻撃を受けるとシステム全体の権限を奪われる恐れがある．機密VM内でIDSを実行する場合には，メモリデータを取得するためにエージェントが必要になる．

この問題を解決するために，本稿では，Arm CCA[1]で導入されたRealmワールドを用いてIDSを動作させ，ノーマルワールドで動作するシステムを監視するCCAmonitorを提案する．CCAmonitorは，Realmワールド内で動作するRealm仮想マシン（VM）上でIDSを実行し，エージェントを用いずにノーマルワールドのメモリ上に格納されているOSデータを取得して監視を行う．Realm VMはノーマルワールドから隔離されているため，ノーマルワールド内の侵入者からIDSへの攻撃を防ぐことができる．また，Realm VMでは既存のOSやアプリケーションを動作させることができるため，高度なIDSを実行するのに適している．さらに，IDSはRealm VMに隔離されるため，攻撃を受けたIDSがシステムメモリやデバイスに直接アクセスすることはできない．

Realm VM内のIDSはRealm Service Interface（RSI）を用いてRealm Management Monitor（RMM）を呼び出し，ノーマルワールドのメモリデータを取得する．RMMはノーマルワールドのメモリ上のページテーブルを参照して，指定されたOSデータの仮想アドレスを物理アドレス

[1] 九州工業大学
Kyushu Institute of Technology

に変換する．そして，対応するノーマルワールドのメモリページをマッピングし，Realm VM のメモリにコピーする．LLView [2] を用いて，監視対象システムの proc ファイルシステムによって提供されるシステム情報を取得する IDS を開発した．この IDS を用いて，ノーマルワールド内と同じシステム情報を取得できることを確認し，IDS の実行性能およびシステム性能に与える影響を測定した．

以下，2 章で TEE を用いた IDS の保護の問題点について述べる．3 章で CCA の Realm VM を用いて IDS を安全に実行する CCAmonitor を提案し，4 章で CCAmonitor の実装について説明する．5 章で CCAmonitor のセキュリティについての分析を行い，6 章で CCAmonitor の動作確認と従来手法との性能比較を行った実験について述べる．7 章で関連研究に触れ，8 章で本稿をまとめる．

2. TEE を用いた IDS の保護

インターネットに接続されたシステムの脆弱性を完全に取り除くことは難しいため，IDS を用いて攻撃を検知する必要がある．近年，シグネチャやルールに基づく従来の IDS だけでなく，機械学習や深層学習を用いて通信ログ，システムコール，プロセス情報などを解析する IDS も用いられるようになっている．このような IDS は，複雑な攻撃パターンや未知の攻撃を検知できる可能性がある一方で，多くの計算資源やメモリを必要とし，複数のプロセスで構成されたり，多数の外部ライブラリに依存したりする場合もある．そのため，このような高度な IDS を実行することができる環境は限られる．一般に，監視対象システムはこのような IDS を動作させるのに適しているが，攻撃者にシステム内に侵入された場合，IDS が停止されたり，改ざんされたりする恐れがある．

IDS を監視対象システムの OS カーネル内で動作させることで，ユーザ空間の攻撃者から保護することができる [3]．しかし，カーネルは高度な IDS を実行するのには適しておらず，OS に脆弱性があった場合には，カーネル内の IDS でさえ攻撃を受ける可能性がある．OS から隔離して IDS を実行する方法として，System Management Mode（SMM）を用いる手法がある [4]．SMM は Intel や AMD のプロセッサが提供する動作モードの一つであり，BIOS 内でのみ使用できる．SMM で IDS を実行することで，OS に侵入した攻撃者から IDS を保護できる．しかし，BIOS も高度な IDS を実行するのには向いていない．また，SMM でプログラムを実行している間はシステム全体が停止するため，監視のたびにシステム性能に影響を及ぼす．SMM を用いてメモリデータを送信し，リモートホストで IDS を実行する手法 [5] もあるが，IDS を安全に実行できるようにする必要がある．

そこで，プロセッサが提供している隔離実行環境（TEE）を用いて IDS を安全に実行する手法が提案されている．Intel SGX を用いることで，アプリケーション内に作成されるエンクレイヴと呼ばれる保護領域で IDS を実行することができる．エンクレイヴ内のメモリは暗号化され，整合性が保証されるため，OS を信頼せずに IDS を安全に実行できる．S-NFV [6] や SEC-IDS [7] はネットワークベース IDS の実行を可能にしており，SSdetector [8] はホストベース IDS を実行することができる．しかし，エンクレイヴはプロセスの一部を保護するための仕組みであり，複数のプロセスを保護する場合にはエンクレイヴを複数実行する [9] か，プロセスの代わりにスレッドを用いる必要がある [10]．また，エンクレイヴ内の IDS はシステムメモリやデバイスに直接アクセスすることはできないため，OS や BIOS などの中で動作するエージェントを用いる必要がある．

Keyspector [11] は，RISC-V Keystone を用いることでエンクレイヴ内の IDS がシステムメモリに直接アクセスすることを可能にしている．通常，Keystone のエンクレイヴは監視対象システムのメモリにアクセスできないが，Keyspector ではセキュリティモニタにおけるコンテキスト切り替え時に RISC-V Physical Memory Protection（PMP）の設定を変更することにより，IDS が動作するエンクレイヴだけにシステムメモリの読み出しを許可する．Keystone のエンクレイヴ内では，アプリケーションだけでなく Eyrie ランタイムや seL4 などの軽量 OS を動かすことができるが，通常の OS とは大きく異なるため，動作させることができる IDS には制約がある．

Arm TrustZone を用いて IDS を安全に実行することもできる．TrustZone を用いる場合，セキュアワールドと呼ばれる実行環境で IDS を実行し，ノーマルワールドで動作するシステムを監視する．例えば，ITZ ライブラリ [12] は，セキュアワールドからノーマルワールドのシステムメモリ上の OS データに直接アクセスする IDS の開発を可能にする．また，LKRDet [13] や Trusted Monitor [14] は，セキュアワールドで CPU の性能カウンタを用いた監視を行う．セキュアワールドでは Trusted OS が動作し，複数の Trusted Application（TA）を実行することができる．しかし，Trusted OS は通常の OS と比べると機能が制限されており，TA として動かせるアプリケーションも限られる．また，Trusted OS に脆弱性があると，セキュアワールドの高い権限が攻撃者に奪われ，システムメモリやデバイスにアクセスされる危険性がある．

機密 VM を用いて IDS を安全に実行する手法も提案されている [15], [16]．機密 VM では既存の汎用 OS，ライブラリ，アプリケーションを動作させることができるため，複数プロセスを用いた高度な IDS を実行するのも容易である．これまでに提案されている手法では，監視対象 VM 内で動作するエージェントを介して監視対象の機密 VM の内部情報を取得する．しかし，エージェントを保護する必要

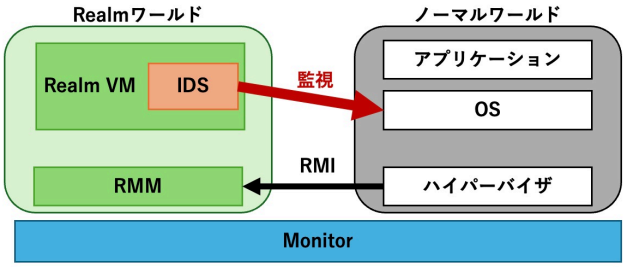

図 1　CCAmonitor のシステム構成

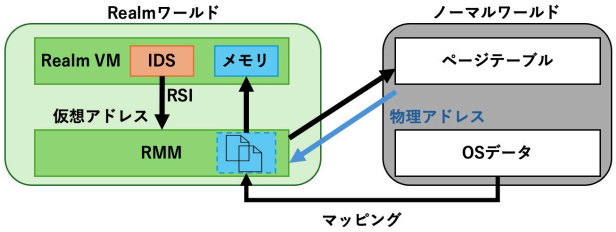

図 2　OS データの取得

があるため，ホスト自身を監視対象とする場合には，ホストに侵入された際にエージェントを安全に動作させ続けることが難しい．また，エージェント経由で情報を取得するとオーバヘッドにより監視性能が低下する．

3. CCAmonitor

本稿では，Arm CCA [1] における機密 VM である Realm VM を用いて，エージェントなしで IDS を安全に実行する CCAmonitor を提案する．CCAmonitor は，CCA で新たに導入された Realm ワールド内で動作する Realm VM で IDS を隔離実行し，ノーマルワールド内で動作するシステムを監視する．従来の VM と同様に，Realm VM では既存の OS，ライブラリ，アプリケーションを動作させることができる．Realm ワールドはノーマルワールドから隔離されるため，ノーマルワールド内に侵入した攻撃者から IDS を保護することができる．一方，IDS は Realm VM に隔離されるため，IDS が攻撃を受け，ゲスト OS の脆弱性を悪用されたとしても，システムメモリやデバイスに直接アクセスすることはできない．

CCAmonitor のシステム構成を図 1 に示す．Arm プロセッサの Realm Management Extension（RME）により，従来の TrustZone で用いられるノーマルワールドとセキュアワールドに加えて，Realm ワールドとルートワールドが提供される．Realm ワールドにおいて Realm Management Monitor（RMM）が動作し，その上に Realm VM を作成して IDS を実行する．監視対象システムはノーマルワールドで動作し，通常の OS とアプリケーションに加えてハイパーバイザも動作する．ハイパーバイザは Realm Management Interface（RMI）を用いて RMM と通信し，Realm VM の管理を行う．ルートワールドでは Monitor が動作し，各ワールドへの物理メモリの割り当ておよび隔離を行う．

CCAmonitor の脅威モデルは以下の通りである．ハードウェア，Monitor，RMM は信頼し，これらに脆弱性はないものとする．一方，ノーマルワールドで動作するアプリケーション，OS，ハイパーバイザは信頼しない．攻撃者は，ネットワークを介してノーマルワールドで動作する監視対象システムに侵入し，OS やハイパーバイザの脆弱性を悪用して高い権限を取得できるものとする．また，Realm VM 内で動作する IDS には脆弱性が存在する可能性があるものとする．攻撃者は，IDS が監視するノーマルワールドの OS データを改ざんすることにより，IDS の脆弱性を悪用することができる．さらに，悪意を持つ IDS が Realm VM 内で実行されることも想定する．

Realm VM 内の IDS はノーマルワールドのメモリ上の OS データを取得することにより，ノーマルワールド内でエージェントを動作させることなくシステムの監視を行う．Realm ワールドはノーマルワールドのメモリにアクセスすることができるが，Realm VM は直接，ノーマルワールドのメモリにアクセスすることはできない．そこで，CCAmonitor では，図 2 に示すように，Realm VM 内の IDS が Realm Service Interface（RSI）を用いて RMM を呼び出し，RMM 経由でノーマルワールドのメモリデータを取得する．IDS が OS データのアドレスを指定して RMM を呼び出すと，RMM は指定されたアドレスに対応するノーマルワールドのメモリページをマッピングして Realm VM のメモリにコピーする．

その際に，RMM は IDS によって指定された OS の仮想アドレスをノーマルワールドの物理アドレスに変換してアクセスを行う．IDS は仮想アドレスを用いて OS のデータ構造を解析する必要があるのに対して，RMM は物理アドレスでしかノーマルワールドのメモリにアクセスできないためである．そこで，CCAmonitor では，RMM がノーマルワールドのメモリ上に格納されたページテーブルを参照することによってこのアドレス変換を行う．RMM はページテーブルのメモリページをマッピングしてページテーブルエントリにアクセスし，ページテーブルをたどる．

4. 実装

CCAmonitor を RMM と Realm VM 内で動作するゲスト OS の Linux に実装した．

4.1 デバイスファイルへのアクセス

Realm VM 内で実行される IDS プロセスがノーマルワールドのメモリデータを取得できるようにするために，ゲスト OS はデバイスファイル/dev/ns_mem を提供する．このデバイスファイルは Linux カーネルモジュールによって作成される．IDS はデバイスファイルに対して pread システムコールを実行することにより，ノーマルワールドのメモ

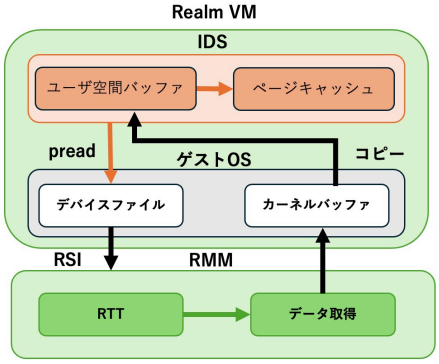

図 3 RMM を介したメモリデータの取得

リデータを取得する．pread に指定するファイルオフセットには，ノーマルワールドの仮想アドレスに対応するページ番号を指定する．ゲスト OS や RMM を呼び出すオーバヘッドを削減するために，pread を用いて複数ページのメモリデータを一括して取得することができる．そのため，pread には必要なサイズのバッファと一括取得するデータサイズを指定する．

IDS は pread を用いて取得したメモリデータをキャッシュに保存してアクセスする．このキャッシュは，4KB のページ境界にそろえた仮想アドレスをキーとし，そのページのメモリデータを値とする．同じページ内のデータが再度参照された場合には，pread を実行せずにキャッシュ上のデータを利用する．図 3 にメモリデータの取得の流れを示す．

4.2 RMM の呼び出し

ゲスト OS は RSI を用いて Realm VM の下で動作する RMM を呼び出す．RSI 呼び出しを用いてノーマルワールドのメモリデータを取得するためのバッファとして，IDS によって指定されたサイズと同じサイズの，物理メモリ上で連続するカーネルメモリを確保する．このカーネルメモリの仮想アドレスを Realm VM の中間物理アドレスへ変換して RSI 呼び出しに指定する．また，読み出しを行うノーマルワールドの先頭の仮想ページ番号と読み出すサイズも指定する．RSI 呼び出しが成功すると，ゲスト OS はカーネルバッファに格納されているノーマルワールドのメモリデータを IDS によって指定されたユーザ空間のバッファへコピーする．

4.3 Realm VM のメモリのマッピング

RMM は Realm VM から渡されたカーネルバッファにアクセスするために，RSI 呼び出しで受け取った Realm VM の中間物理アドレスを RMM からアクセス可能なアドレスへ変換する．RMM はまず，Realm VM ごとに用意される Realm Translation Table（RTT）を用いて中間物理アドレスを物理アドレスへ変換し，対応するグラニュールを取得する．グラニュールは CCA において物理メモリを管理するために用いられる単位である．そして，そのグラニュールを RMM の仮想アドレス空間に一時的にマッピングする．RMM はそのメモリ領域にノーマルワールドから取得したデータを書き込んだ後，グラニュールをアンマップする．複数ページのメモリデータを一括取得する場合，この処理を要求されたページ数だけ繰り返す．

4.4 ノーマルワールドの仮想アドレスの変換

RMM はノーマルワールドのメモリ上に格納されているページテーブルをたどり，IDS によって指定された仮想アドレスを物理アドレスへ変換する．RMM はノーマルワールドから Realm ワールドに切り替わる際に保存されるレジスタ値を参照し，TCR_EL1 から仮想アドレス幅とページテーブルのグラニュールサイズ，TTBR1_EL1 からカーネル空間に用いられるページテーブルの物理アドレスを取得する．

仮想アドレス幅からページテーブルウォークの開始レベルを決定し，各レベルのインデックスを用いてページテーブルエントリを読み出す．各エントリを読み出す際には，その物理アドレスに対応するグラニュールがノーマルワールドに属することを確認する．エントリが次のレベルのテーブルを指している場合には，その物理アドレスを用いてページテーブルウォークを継続する．一方，1GB または 2MB のブロックを指すエントリ，あるいは 4KB ページを指すエントリに到達した場合には，エントリから得られた物理アドレスと仮想アドレス内のオフセットを組み合わせて最終的な物理アドレスを得る．

4.5 ノーマルワールドのメモリのマッピング

RMM はノーマルワールドの物理アドレスに対応するグラニュールをマッピングすることにより，ページテーブルや OS データが格納されたメモリ領域にアクセスする．CCAmonitor はグラニュールをその都度マッピングするデマンドマッピングと，ノーマルワールドに割り当てられたグラニュールをあらかじめマッピングしておくプリマッピングを提供する．

4.5.1 デマンドマッピング

RMM は物理アドレスを 4KB 境界にそろえて対応するグラニュールを取得し，ノーマルワールドに属することを確認する．グラニュールをデマンドマッピング用に用意されている RMM の仮想アドレスにマッピングする．そして，ページテーブルエントリにアクセスする場合には，8 バイトのページテーブルエントリをコピーする．OS データを取得する場合には，4KB のページのメモリデータを Realm VM から渡されたカーネルバッファにコピーする．コピーが終了するとグラニュールをアンマップする．複数ページを一括取得する場合には，この処理をページごとに

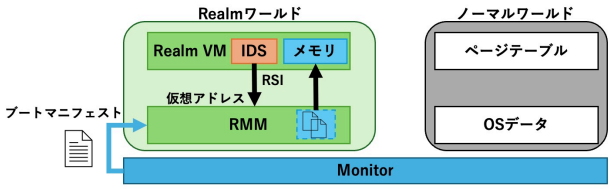

図 4 メモリ全体のプリマッピング

繰り返す.

4.5.2 プリマッピング

OS データ取得時に行うデマンドマッピングのオーバヘッドを削減するために，RMM の起動時にノーマルワールド用のメモリ領域を RMM の仮想アドレス空間にあらかじめマッピングしておくことができる．これにより，アクセスのたびにグラニュールをマッピング・アンマッピングする必要がなくなり，物理アドレスを対応する仮想アドレスに変換するだけでアクセスできる．その一方で，起動時のマッピングに時間がかかり，IDS がアクセスしない領域のマッピング処理は無駄になる．

図 4 にプリマッピングによるデータ取得の流れを示す．Monitor は，RMM との通信用に確保された共有メモリ領域にブートマニフェストを格納し，その先頭物理アドレスを RMM に通知する．RMM はブートマニフェストから，ノーマルワールド用のメモリ領域の先頭物理アドレスとサイズを取得する．RMM はこの領域をステージ 1 ページテーブルへ登録する．RSI 呼び出し時には，目的の物理アドレスとこの領域の先頭物理アドレスとの差をマッピング先の先頭仮想アドレスに加えることにより，アクセス先の仮想アドレスを得る．

4.6 透過的な OS データアクセス

CCAmonitor では LLView [2] を用いて，IDS がノーマルワールドの OS データに透過的にアクセスすることを可能にする．LLView は，Linux カーネルのソースコードを用いて OS データを解析するプログラムを記述するためのフレームワークである．カーネルの構造体，グローバル変数，マクロを利用できるため，IDS をカーネルモジュールに近い形で記述できる．LLView は，IDS のコンパイル時に中間表現においてカーネルのグローバル変数をノーマルワールドの仮想アドレスに置き換える．さらに，OS データを読み出すロード命令の前に，データ取得関数の呼び出しを挿入する．この関数は，対象のページがキャッシュに存在する場合にはキャッシュ上のデータを返し，存在しない場合にはデバイスファイルを介して RMM を呼び出す．これにより，IDS の開発者は RSI 呼び出しを手動で記述する必要がない．

表 1 IDS が取得するシステム情報

ファイルパス	提供されるシステム情報
/proc/sys/kernel/osrelease	Linux カーネルのリリース番号
/proc/uptime	システムの稼働時間
/proc/stat	CPU 時間などのシステム統計情報
/proc/tty/drivers	登録されている TTY ドライバ
/proc/meminfo	メモリの使用状況
/proc/net/tcp	TCP コネクションの情報
/proc/<PID>/auxv	プロセスの補助ベクトル
/proc/<PID>/stat	プロセスの状態および統計情報
/proc/<PID>/status	プロセスの状態とメモリ使用量

4.7 proc ファイルシステムの情報取得

LLView を用いて，Linux の proc ファイルシステムが提供するシステム情報を取得する IDS を実装した．この IDS は，Linux カーネルの proc ファイルシステムのソースコードを用いて，ノーマルワールドのメモリ上にある OS データから疑似ファイルのデータを生成する．現在の実装では，表 1 のシステム情報を取得する．

5. セキュリティ分析

まず，ノーマルワールドから IDS への攻撃について考える．CCAmonitor では，IDS を Realm VM 内で実行するが，Realm VM のメモリは CCA によってノーマルワールドから隔離されている．そのため，ノーマルワールドに侵入して OS やハイパーバイザの権限を取得した攻撃者であっても，IDS のコードやデータを直接読み書きすることはできない．そのため，監視対象システム内で IDS を実行する場合と異なり，攻撃者による IDS の盗聴や改ざんを防ぐことができる．

一方，Realm VM の実行制御はノーマルワールドで動作するハイパーバイザが行うため，ハイパーバイザが攻撃された場合，Realm VM 内の IDS が実行されない可能性がある．この攻撃への対策として，RMM が Realm VM の実行状況を記録し，一定時間実行されていない場合や IDS によるメモリデータの取得が行われない場合に，異常として検出することが考えられる．これにより，Realm VM の停止自体を防ぐことはできないが，監視が正常に行われていないことを検知できる．

次に，IDS からノーマルワールドへの攻撃について考える．IDS は Realm VM 内に隔離されるため，直接ノーマルワールドへの攻撃を行うことはできない．Realm VM 内のゲスト OS の権限を取得したとしても同様である．一方，CCAmonitor では，IDS は RMM を呼び出すことでノーマルワールドの任意のメモリを読み出すことができる．そのため，IDS の脆弱性が悪用された場合や，悪意のある IDS が実行された場合には，監視に必要のないメモリを読み出される恐れがある．この攻撃に対しては，RMM が IDS から指定された仮想アドレスをチェックし，読み出し可能な範囲を監視に必要なカーネルデータ領域に限定することが

```
# ./procfs
/proc/stat:
cpu  11402 0 1726 15620 3 917 30 0 11310 0
cpu0 11402 0 1723 2171 3 667 29 0 11310 0
cpu1 0 0 1 3321 0 235 0 0 0 0
cpu2 0 0 0 3774 0 0 0 0 0 0
cpu3 0 0 0 6353 0 13 0 0 0 0
intr 88902 0 2058 23290 0 0 0 290 0 0 0 0 0 36498 0 0 0 0 0 4 0 0 0 232 89 4 0 0
 0 429 0 0 0 0 0 185 0 687 0 0 25159 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0
ctxt 88754
btime 1770257640
processes 183
procs_running 2
procs_blocked 0
softirq 8841 39 4092 50 154 47 0 236 0 0 4223
```

図 5 取得したシステム情報（一部）

考えられる．これにより，アプリケーションのデータの盗聴を防ぐことができる．

6. 実験

CCAmonitor の有効性を調べるために，開発した IDS を用いて実験を行った．この IDS は，ノーマルワールドのメモリデータを解析し，proc ファイルシステムによって提供されるシステム情報を取得する．比較として，ノーマルワールド内で proc ファイルシステムを読み出す従来 IDS を用いた．Arm CCA に対応した実機はまだないため，この実験では QEMU 8.2.2 上で CCA のエミュレーション環境 [17] を用いた．実験に用いたマシンおよび，エミュレーション環境上のノーマルワールド，Realm VM のスペックは表 2 の通りである．

6.1 IDS の動作確認

CCAmonitor を用いて，Realm VM 内で IDS を実行し，ノーマルワールドのシステム情報の取得を行った．IDS が取得したシステム情報の一部を図 5 に示す．取得した情報をノーマルワールド内で実行した従来 IDS の出力と比較したところ，同じ情報が取得できていることを確認した．

6.2 システム情報の取得時間

CCAmonitor を用いて，ノーマルワールドの proc ファイルシステムによって提供されるシステム情報を取得するのにかかる時間を測定した．まず，RMM においてデマンドマッピングを行う場合に，この IDS が一括取得するページ数の最適値を調べるために，一括取得するページ数を変えながら実行を行った．取得時間を 10 回測定した時の平均値を図 6 に示す．この結果より，IDS が一括取得するページ数を増やすと取得時間が短くなるが，2 ページで最小となり，それ以上に増やすと取得時間が長くなることが分かった．

この時の RSI 呼び出し回数は図 7 に示すようになり，一括取得するページ数を増やしていくと 4 ページまでは呼び出し回数が大きく減少したが，それ以上に増やしても少しずつしか減少しなかった．これは，連続するメモリの

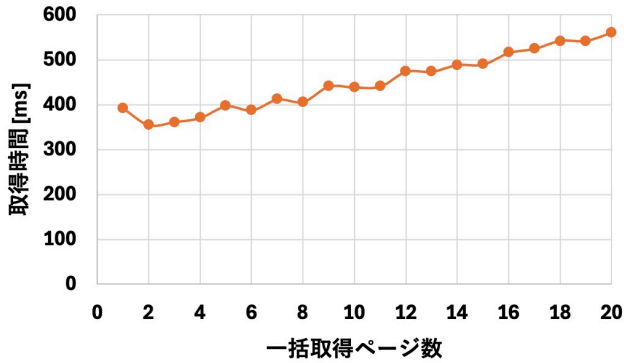

図 6 一括取得するページ数を変更した時の取得時間

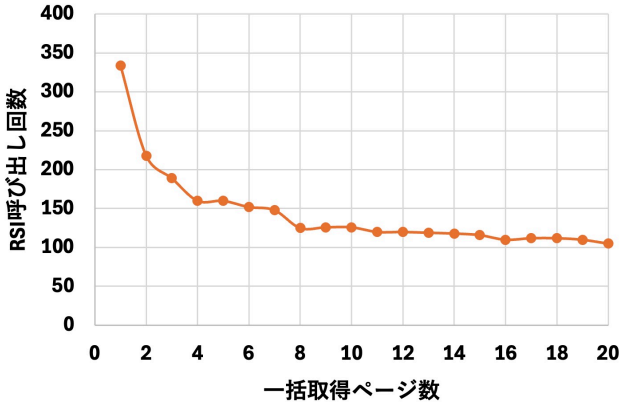

図 7 RSI 呼び出し回数

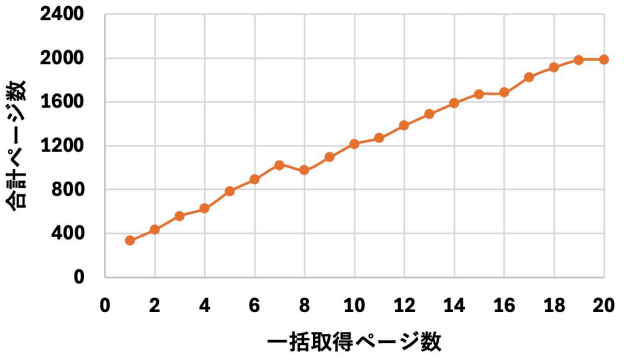

図 8 取得した合計ページ数

データを一括で取得しても，IDS によってアクセスされないページが多くなるためだと考えられる．IDS が必要とするページ数は 1 ページずつ取得した時の 334 ページであるが，一括取得するページ数を増やしていくと図 8 のように，取得する合計ページ数が大幅に増加していく．そのため，一括取得するページ数が 2 ページを超えると，RSI 呼び出し回数の減少によるオーバヘッド削減よりも，取得するページ数の増加によるオーバヘッドの方が大きくなったと考えられる．

オーバヘッドの詳細を調べるために，RSI 呼び出し 1 回あたりの処理時間の内訳を測定した．測定結果を図 9 に示す．Realm VM 内で IDS がデバイスファイルからメモリ

表 2　実験環境

	ホスト	ノーマルワールド	Realm VM
CPU	Intel Core i7-14700	Arm CPU（4 コア）	仮想 CPU（2 コア）
メモリ	32GB	8GB	512MB
OS	Linux 6.14	Linux 6.15	Linux 6.15

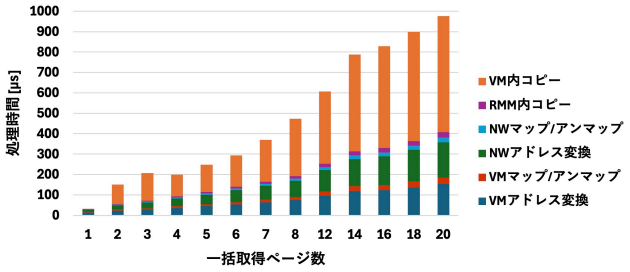

図 9　RSI 呼び出し 1 回あたりの処理時間の内訳

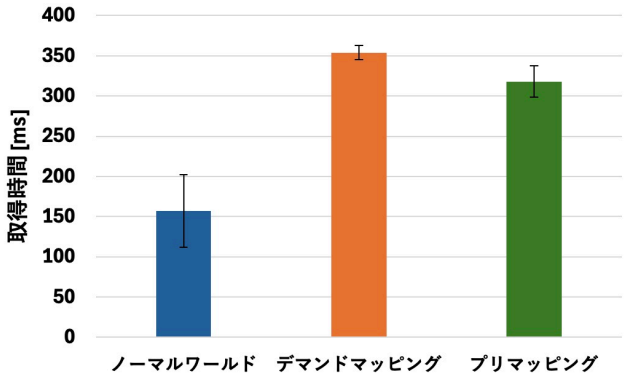

図 10　取得時間

データをコピーするのにかかる時間が大きな割合を占めていることが分かる．その次に，RMM 内でのノーマルワールド（NW）と Realm VM のメモリのアドレス変換に時間がかかっていた．

次に，CCAmonitor においてデマンドマッピングを行った場合とプリマッピングを行った場合，および，ノーマルワールド内で従来 IDS を実行した場合の取得時間の比較を行った．CCAmonitor を用いる場合には，IDS は 2 ページを一括取得した．測定結果は図 10 に示すようになり，従来 IDS と比べて CCAmonitor は 2.0〜2.3 倍の取得時間がかかった．これは Realm VM 内の IDS がノーマルワールドのメモリデータを安全に取得してシステム情報を解析するオーバヘッドのためである．

プリマッピングを行うと RSI 呼び出し時に RMM でのノーマルワールドのメモリのマッピングが不要になるが，デマンドマッピングを行う場合と比べた性能向上は 11.3% であった．性能向上が小さいのは，図 9 に示すように，RMM においてノーマルワールドのメモリのマッピング・アンマッピングを行うのにかかる時間が相対的に短いことが原因である．

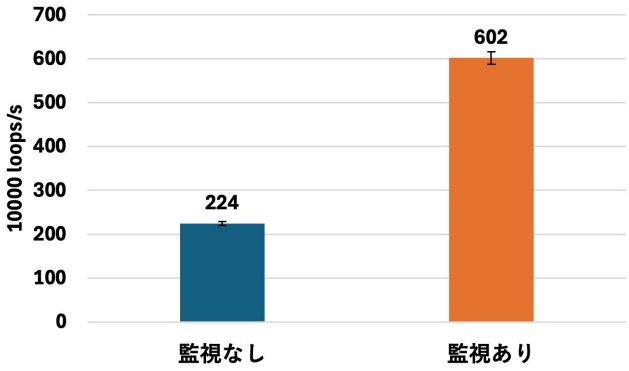

図 11　Dhrystone の実行結果

6.3　システム性能への影響

ノーマルワールド内で Dhrystone ベンチマークを実行し，Realm VM 内で IDS を動作させた時と動作させなかった時の CPU 性能の比較を行った．この実験では，IDS は 10ms 間隔でシステム情報の取得を行うようにした．ベンチマーク結果は図 11 に示すようになり，IDS を実行することにより Dhrystone のスコアが上昇した．この原因として，Realm VM での IDS の実行がノーマルワールドのスケジューリング等に影響を及ぼした可能性や，エミュレーション環境を用いたためである可能性が考えられる．

7.　関連研究

SGmonitor [18] や SCwatcher [19] は，Intel SGX を用いて IDS を保護することで，VM を安全に監視するシステムである．SGX のエンクレイヴ内で IDS を実行することにより，IDS の改ざんや，IDS が取得した情報の漏えいを防ぐ．エンクレイヴ内の IDS はハイパーバイザ経由で VM のメモリデータを取得するため，ハイパーバイザを信頼する必要がある．SCwatcher [19] は，ライブラリ OS の Occlum を用いて疑似的に複数プロセスの実行を可能にし，エンクレイヴ内の IDS に OS の標準インタフェースを提供する．仮想的な proc ファイルシステムを用いて監視対象 VM のシステム情報を提供することで，既存の IDS を変更せずに実行することができる．

00SEVen [20] は，AMD SEV-SNP によって保護された機密 VM をリモートホストから監視することを可能にするシステムである．機密 VM のメモリデータを取得できるようにするために，VM 内でエージェントを動作させてメモリデータをリモートホストに転送する．監視対象 VM 内のエージェントを保護するために，SEV-SNP の VMPL を

用いて，VM 内の最も高い特権レベルでエージェントを実行する．リモートホストで IDS を動作させることにより，高度な IDS が実行可能であるが，通信を保護する必要がある．また，リモートホスト上で実行される IDS を保護する必要もある．クラウド外の信頼できる環境にあるホスト上で IDS を実行しようとすると，通信遅延が大きくなる．

SEVmonitor [15] は，AMD SEV を用いて保護された機密 VM を別の機密 VM から監視するシステムである．00SEVen とは異なり，監視対象システムをコンテナや内部 VM に隔離し，その外側のゲスト OS や内部ハイパーバイザの中でエージェントを安全に動作させる．機密 VM で IDS を実行するため，高度な IDS の実行が可能である．IDS とエージェント間の通信を保護する必要があるが，クラウド内で IDS を実行することができるため，通信遅延を抑えることができる．監視対象 VM と同一ホスト上で IDS を実行する場合には，共有メモリを用いてさらに通信遅延を小さくすることができる．

TVMmonitor [16] は，RISC-V CoVE によって保護された機密 VM を別の機密 VM から監視するシステムである．監視対象 VM のゲスト OS の中でエージェントを動作させ，機密 VM 間で確立された共有メモリを用いて，OS データを直接 IDS に転送する．さらに，IDS はエージェントを用いずに，機密 VM の下で動作する，信頼できる TSM 経由で監視対象 VM のメモリデータをコピーして取得することもできる．しかし，アドレス変換とメモリコピーのオーバヘッドが大きい．

8. まとめ

本稿では，Arm CCA で導入された Realm ワールドを用いて Realm VM 上で IDS を安全に実行し，ノーマルワールドで動作するシステムを監視することを可能にする CCAmonitor を提案した．IDS は Realm VM の下で動作する RMM を介してノーマルワールドのメモリデータを取得し，その中の OS データの監視を行う．実験により，CCAmonitor を用いてノーマルワールドの proc ファイルシステムが提供するシステム情報と同じ情報を取得できることを確認し，IDS の実行性能およびシステム性能に与える影響を測定した．

今後の課題は，Realm VM 内の IDS プロセスがノーマルワールドのメモリを直接マッピングしてアクセスできるようにすることである．現在の実装では，IDS はゲスト OS 経由で RMM を呼び出してノーマルワールドのメモリデータを取得しているため，呼び出しやメモリコピーに伴うオーバヘッドが大きい．また，RMM において Realm VM が正常に動作しているかどうかをチェックできる仕組みが必要である．Realm VM はノーマルワールドのハイパーバイザによって実行制御が行われるため，ハイパーバイザが攻撃を受けた場合，IDS が動作する Realm VM を停止される可能性がある．

謝辞 本研究は，JST 経済安全保障重要技術育成プログラム【JPMJKP24U4】の支援を受けたものである．

参考文献

[1] Arm: Arm Confidential Compute Architecture. 入手先 ⟨https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture⟩ (2026.06.19).
[2] Ozaki, Y., Kanamoto, S., Yamamoto, H. and Kourai, K.: Detecting System Failures with GPUs and LLVM, *APSys '19: Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, pp. 47–53 (2019).
[3] Breitenbacher, D., Homoliak, I., Aung, Y. L., Tippenhauer, N. O. and Elovici, Y.: HADES-IoT: A Practical Host-Based Anomaly Detection System for IoT Devices, *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pp. 479–484 (2019).
[4] Zhang, F., Leach, K., Sun, K. and Stavrou, A.: SPECTRE: A Dependable Introspection Framework via System Management Mode, *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 1–12 (2013).
[5] Zhang, F., Wang, J., Sun, K. and Stavrou, A.: HyperCheck: A Hardware-Assisted Integrity Monitor, *IEEE Transactions on Dependable and Secure Computing*, Vol. 11, No. 4, pp. 332–344 (2014).
[6] Shih, M.-W., Kumar, M., Kim, T. and Gavrilovska, A.: S-NFV: Securing NFV states by using SGX, *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pp. 45–48 (2016).
[7] Kuvaiskii, D., Chakrabarti, S. and Vij, M.: Snort Intrusion Detection System with Intel Software Guard Extension (Intel SGX) (2018). arXiv:1802.00508.
[8] Koga, Y. and Kourai, K.: SSdetector: Secure and Manageable Host-based IDS with SGX and SMM, *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 539–548 (2023).
[9] Tsai, C.-C., Porter, D. E. and Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX, *2017 USENIX Annual Technical Conference*, pp. 645–658 (2017).
[10] Shen, Y., Tian, H., Chen, Y., Chen, K., Wang, R., Xu, Y., Xia, Y. and Yan, S.: Occlum: Secure and Efficient Multitasking Inside a Single Enclave of Intel SGX, *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 955–970 (2020).
[11] Iwano, T. and Kourai, K.: Keyspector: Secure Monitoring of IoT Devices Using RISC-V Keystone, *2025 IEEE 30th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 102–112 (2025).
[12] Guerra, M., Taubmann, B., Reiser, H. P., Yalew, S. and Correia, M.: Introspection for ARM TrustZone with the ITZ Library, *2018 IEEE International Conference on Software Quality, Reliability and Security*, pp. 123–134 (2018).
[13] Jiang, X., Lora, M. and Chattopadhyay, S.: Efficient and Trusted Detection of Rootkit in IoT Devices via Offline Profiling and Online Monitoring, *Proc. 30th ACM Great*

Lakes Symp. on VLSI, pp. 433–438 (2020).
[14] Eichler, C., Röckl, J., Jung, B., Schlenk, R., Müller, T. and Hönig, T.: Profiling with trust: system monitoring from trusted execution environments, *Design Automation for Embedded Systems*, Vol. 28, No. 1, pp. 23–44 (2024).
[15] Nono, T. and Kourai, K.: Secure Monitoring of Confidential VMs with Isolated Agents, *Proceedings of the 2025 IEEE/ACM 18th International Conference on Utility and Cloud Computing (UCC '25)* (2025).
[16] 梶原悠大, 光來健一: RISC-V CoVE を用いた Confidential VM の効率的かつ柔軟な監視機構, *ComSys* (2025).
[17] Poirier, M.: Building an RME Stack for QEMU (2024). 入手先 ⟨https://linaro.atlassian.net/wiki/pages/viewpage.action?pageId=29275783169⟩ (2026.06.27).
[18] Nakano, T. and Kourai, K.: Secure Offloading of Intrusion Detection Systems from VMs with Intel SGX, *Proceedings of the 2021 IEEE 14th International Conference on Cloud Computing (CLOUD '21)*, pp. 297–303 (2021).
[19] Kawamura, T. and Kourai, K.: Secure Offloading of User-level IDS with VM-compatible OS Emulation Layers for Intel SGX, *Proc. IEEE Int. Conf. Cloud Computing (CLOUD '22)* (2022).
[20] Schwarz, F. and Rossow, C.: 00SEVen – Re-enabling Virtual Machine Forensics: Introspecting Confidential VMs Using Privileged in-VM Agents, *Proceedings of the 33rd USENIX Security Symposium C*, pp. 1651–1668 (2024).