RISC-V Keystoneを用いたIoT機器の安全な監視

岩野 空仁[1]　光来 健一[1]

概要：IoT 機器はインターネットからの攻撃を受けやすいため，侵入検知システム (IDS) を動作させて監視を行う必要があるが，監視対象システム内で動作する IDS は無効化されるリスクがある．そこで，Intel SGX や Arm TrustZone などの隔離実行環境 (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，監視対象システムのメモリにアクセスするオーバヘッドが大きかったり，IDS の権限が高すぎたりするという問題がある．本稿では，RISC-V プロセッサの TEE である Keystone を用いてエンクレイヴ内で IDS を安全に実行するシステム Keyspector を提案する．Keyspector は IDS を動作させるエンクレイヴだけが監視対象システムのメモリを共有できるようにし，IDS がシステムメモリに格納された OS データを直接取得することを可能にする．Keyspector をセキュリティモニタと Eyrie ランタイムに実装し，proc ファイルシステムの情報を取得する IDS を開発して有効性を確かめる実験を行った．

1. はじめに

近年，あらゆるモノがインターネットに接続される Internet of Things (IoT) が急速に普及しており，2027 年までに IoT 機器は 572.6 億台まで増加すると見込まれている [1]．IoT では，従来，インターネットに接続されていなかった様々な機器がネットワークを通じてサーバなどに接続され，情報交換を行う．そのため，IoT 機器は外部からの攻撃を受けるリスクが高い．実際に，インターネット上で到達可能かつ未使用の IP アドレス空間であるダークネットで観測されたパケットのうち，30%以上が IoT 機器に関連したサイバー攻撃関連の通信であった [2]．IoT 機器への攻撃の例として，IoT 機器に侵入されて分散サービス妨害 (DDoS) 攻撃などに利用されたり [3]，機器内で動作していたプログラムが改ざんされたり [4], [5] する事例が発生している．

IoT 機器への侵入を検知するためには侵入検知システム (IDS) を動作させて監視を行う必要があるが，監視対象システム内で動作する IDS は侵入者に無効化されるリスクがある．そこで，最近の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．例えば，Intel SGX のエンクレイヴと呼ばれる保護領域で IDS を実行する手法 [6] では，IDS が監視対象システムのメモリにアクセスしてメモリ上の OS データを監視する．しかし，エンクレイヴ内からはシステムメモリに直接アクセスできないため，メモリデータを取得するオーバヘッドが大きい．一方，Arm TrustZone のセキュアワールドで IDS を実行する手法 [7] では，IDS がシステムメモリに直接アクセスすることができる．しかし，セキュアワールドは IDS が必要とするよりも高い権限を持つため，IDS の実行にはリスクが伴う．

この問題を解決するために，本稿では，RISC-V プロセッサの TEE である Keystone [8] を用いて IDS を安全に実行する Keyspector を提案する．RISC-V は近年，注目されているオープンソースの命令セットアーキテクチャである．Keyspector はエンクレイヴ内で IDS を実行し，IDS がシステムメモリに直接アクセスすることを可能にする．そのために，セキュリティモニタが Physical Memory Protection (PMP) と呼ばれる RISC-V のメモリ保護機構を用いて，エンクレイヴにシステムメモリを共有させる．エンクレイヴの悪用を防ぐために，エンクレイヴ内の IDS のハッシュ値が事前に登録されたものと一致する場合にのみ，システムメモリへのアクセスを許可する．そして，エンクレイヴ内で動作する軽量 OS（ランタイム）がシステムメモリを IDS のアドレス空間にマッピングすることで，IDS からシステムメモリへのアクセスを実現する．

Keyspector をセキュリティモニタと Eyrie ランタイムに実装し，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 はシステムメモリ上にあるページテーブルを用いて，OS データの仮想アドレスを物理アドレスに変換する．ページテーブルのアドレスは IDS がランタイム経由でセキュリティモニタを呼び出すことにより，監視対象システムが使用している CPU レジスタの値から取得す

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

る．また，LLView [9] を RISC-V に対応させることにより，IDS が OS のソースコードを用いて監視対象システムの OS データに透過的にアクセスすることを可能にしている．この IDS を用いて実験を行ったところ，監視対象システム内と同じシステム情報を取得できることを確認した．また，監視対象システム内で proc ファイルシステムを読み出す従来手法の取得時間も測定し，比較を行った．

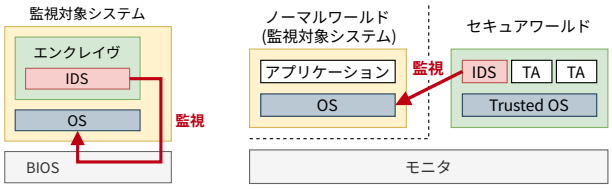
以下，2 章では IoT 機器の監視における問題点について述べる．3 章では RISC-V の Keystone を用いて IDS を安全に実行し，監視対象システムのメモリデータを効率よく取得することを可能にする Keyspector を提案する．4 章では Keyspector の実装について説明し，5 章では Keyspector の動作確認と監視対象システム内で監視した場合との比較を行った実験について述べる．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. IoT 機器の監視

IoT 機器はインターネットに接続されるため，外部からの攻撃を受けるリスクが高い．攻撃者の侵入の糸口となる IoT 機器の脆弱性を完全に取り除くのは難しいため，侵入検知システム (IDS) を用いて攻撃を検知する必要がある．IDS は IoT 機器内のシステムの異常を監視し，管理者に対して通知を行う．しかし，監視対象システム内で IDS を安全に実行するのは難しい．なぜなら，システム内に侵入された場合，IDS を改ざんされる恐れがあるためである．IDS を OS 内で動作させることによって保護することができるが，OS に脆弱性があった場合には OS 内で動作する IDS でさえ攻撃を受ける可能性がある．

そこで，IDS を安全に実行できるようにするために，プロセッサが提供する隔離実行環境 (TEE) が用いられている．TEE の一つである Intel SGX は，アプリケーション内にエンクレイヴと呼ばれる保護領域を作成することができる．プログラム実行開始時には SGX によって電子署名が検査されるため，攻撃者によって改ざんされたプログラムは実行されない．また，SGX によってエンクレイヴのメモリの整合性が保証されるため，攻撃者による実行中のプログラムの改ざんを防ぐことができる．加えて，エンクレイヴのメモリは暗号化されるため，エンクレイヴ内のデータが攻撃者に盗聴されることもない．このようにして SGX のエンクレイヴ内のプログラムは，OS を信頼せずとも CPU の保護により安全に実行できる．

SGX を用いる SSdetector [6] は図 1(a) のように，エンクレイヴ内で IDS を実行することで，IDS の盗聴や改ざんを防ぐ．IDS はシステムメモリを参照して監視を行うが，SGX のエンクレイヴ内からはシステムメモリに直接アクセスすることができない．そのため，CPU の動作モードの一つであるシステムマネジメントモード (SMM) で動作する BIOS 内のプログラムを呼び出して安全にメモリデータの取得を行う．その際に，メモリデータの暗号化や整合性検査を行って OS などによる盗聴・改ざんを防ぐ．しかし，この BIOS の呼び出しとデータの保護のために，メモリデータ取得のオーバヘッドが大きい．また，SMM でのプログラムの実行中はシステム全体が停止するため，システム性能への影響も大きい．それに加えて，現在，SGX をサポートしているのはサーバ向け CPU のみであるため，IoT 機器では利用することができない．



(a) SGX　　(b) TrustZone

図 1　TEE を用いた IDS

IoT 機器でよく用いられている Arm プロセッサでは TrustZone と呼ばれる TEE が提供されており，ワールドと呼ばれる実行環境が 2 つ用意される．ノーマルワールドでは Android や Linux などの一般的な OS が動作する．一方，セキュアワールドでは，ノーマルワールドから隔離したい Trusted Application (TA) を動作させる．セキュアワールドではノーマルワールドの OS とは別に Trusted OS が動作する．これら 2 つのワールド間の切り替えはモニタによって行われ，システムリソースは厳密に分離される．そのため，ノーマルワールドからはセキュアワールド用に確保されたリソースにアクセスすることはできない．

ITZ ライブラリ [7] は図 1(b) のように，TrustZone の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にアクセスすることを可能にする．このライブラリを用いることで IDS の開発が可能である．例えば，カーネルメモリのハッシュ値を計算して正常な状態のハッシュ値と比較することで整合性検査を行ったり，カーネル変数の値を取得したりすることができる．しかし，セキュアワールドはデバイスへのアクセスなど，IDS が必要とする以上の権限を持っており，IDS が攻撃を受けるとセキュアワールドの高い権限を奪われる恐れがある．実際，TA や Trusted OS にはさまざまな脆弱性があることが報告されている [10]．そのことから，IDS の開発には細心の注意を払う必要があり，多くの機器ではユーザが開発した TA を実行することを許していない．

3. Keyspector

本稿では，RISC-V プロセッサの TEE である Keystone を用いて IDS を安全に実行するシステム Keyspector を提案する．RISC-V は近年，注目されているオープンソースの命令セットアーキテクチャであり，IoT 機器での利用が

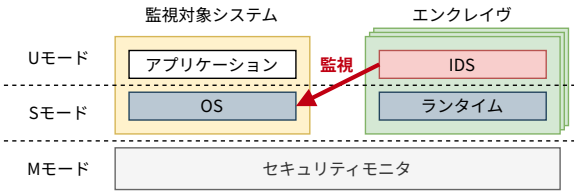


図 2 Keyspector のシステム構成

見込まれている．Keyspector のシステム構成を図 2 に示す．Keypector はエンクレイヴ内で IDS を実行することにより，IDS の盗聴や改ざんを防ぐ．エンクレイヴと監視対象システムは M モードで動作するセキュリティモニタ上で実行され，互いに強固に隔離される．Keyspector では，ハードウェアおよびセキュリティモニタを信頼できるものとし，脆弱性はないものとする．攻撃としては，外部の攻撃者によるネットワーク経由での監視対象システムへの侵入を想定する．

Keyspector はエンクレイヴ内の IDS が監視対象システムのメモリを直接参照することを可能にする．Keystone では，セキュリティモニタが Physical Memory Protection (PMP) と呼ばれる RISC-V のメモリ保護機構を用いてシステムとエンクレイヴのメモリを論理的に分離しており，エンクレイヴからシステムのメモリには一部を除いてアクセスすることはできない．Keyspector は PMP においてアクセス可能なメモリの範囲や権限を変更することにより，エンクレイヴがシステムメモリ全体を共有できるようにする．エンクレイヴ内では軽量 OS（ランタイム）も動作するため，IDS がシステムメモリに直接アクセスできるようにするために，ランタイムがシステムメモリを IDS のメモリアドレス空間にマッピングする．

IDS はマッピングされたシステムメモリにアクセスすることで，監視対象システムの OS データの監視を行う．そのために，システムメモリ上にある OS のページテーブルを用いて，OS データの仮想アドレスを物理アドレスに変換する．RISC-V ではページテーブルのアドレスは CPU レジスタに格納されているため，IDS はランタイム経由でセキュリティモニタを呼び出すことにより，システムからエンクレイヴへの切り替え時に保存されたレジスタ値を取得する．Keyspector はこのようなアドレス変換を意識せずに IDS を開発できるようにするために，IDS のコンパイル時にアドレス変換の処理を埋め込むことで透過的にアドレス変換が行われるようにする．これにより，OS のソースコードを利用して，IDS をカーネルモジュールのように記述することができる．

Keyspector のセキュリティモニタは，IDS が実行されているエンクレイヴに対してのみシステムメモリへのアクセスを許可する．これは，すべてのエンクレイヴがシステムメモリにアクセスできると，システムに侵入した攻撃者がエンクレイヴを実行してシステムメモリを盗聴できてしまうためである．セキュリティモニタはアテステーションのために，エンクレイヴ内部で実行されるソフトウェアのハッシュ値を計算しているため，それがあらかじめ登録された IDS のハッシュ値と一致する場合にのみシステムメモリの共有を行う．別の方法として，リモートアテステーションを行う際に検証サーバで IDS のハッシュ値と一致するかどうかを判定し，その結果をセキュリティモニタに通知することも考えられる．

Keyspector のセキュリティモニタはエンクレイヴの実行状態を監視することで，監視対象システム内の侵入者による IDS への攻撃を検知する．エンクレイヴ内で動作する IDS を実行するには，監視対象システム内で IDS プロセスを実行してエンクレイヴを作成する必要がある．そのため，侵入者に管理者権限を奪われてしまうと，IDS プロセスを不正に停止されて，IDS を無効化される恐れがある．セキュリティモニタは定期的に IDS が動作しているエンクレイヴの存在を確認することで，IDS プロセスが不正に終了させられたことを検知することができる．また，コンテキストスイッチを基に IDS が動作しているエンクレイヴの実行時間を監視することで，IDS が動いているかどうかを確認することができる．

SGX とは異なり，Keyspector のエンクレイヴはシステムメモリ全体に直接アクセスすることができるため，取得するメモリデータを保護するための暗号化や整合性検査は不要である．SGX のエンクレイヴはそれが動作しているプロセスのメモリにしか直接アクセスできないため，システムメモリにアクセスするには BIOS 等を介する必要があり，データ保護が必要となる．また，Keyspector ではセキュリティモニタでエンクレイヴの実行状態を監視することができるが，SGX ではこのようなプログラム可能かつ信頼できる特権ソフトウェアはない．一方，TrustZone のセキュアワールドと比べて Keyspector のエンクレイヴの権限は低いため，より安全に IDS を実行することができる．エンクレイヴの権限が奪われたとしても，攻撃者はシステムメモリを盗聴することしかできない．その代わり，Keyspector のエンクレイヴはより容易に無効化することができるが，セキュリティモニタで無効化を検知することができる．

4. 実装

Keyspector を Keystone のセキュリティモニタと Eyrie ランタイムに実装した．

4.1 システムメモリの共有

PMP はソフトウェアがアクセス可能な物理アドレスの範囲にアクセス権限を設定するためのメモリ保護機構であり，pmpcfg レジスタと pmpaddr レジスタで構成される．

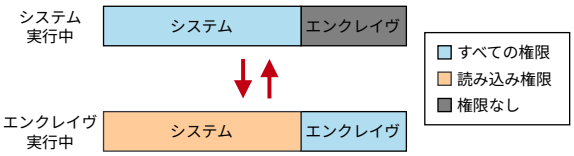


図 3 物理メモリのアクセス権限

pmpcfg レジスタでは各物理メモリ領域に対して読み込み，書き込み，実行のアクセス権限の指定などを行うことができる．pmpaddr レジスタではアクセス権限の指定を適用する物理メモリ領域の先頭アドレスとサイズの指定を最小 4 バイトから設定することができる．レジスタにつけられた番号が優先度を示しており，セキュリティモニタが最も高い優先度のレジスタを使い，監視対象システムが最も低い優先度のレジスタを使う．PMP のチェックは特権モードが S モードもしくは U モードであるすべてのアクセスに対して適用される．

Keystone では，システムとエンクレイヴがアクセスできる物理アドレスの範囲とアクセス権限をセキュリティモニタが PMP に設定する．セキュリティモニタはシステムからエンクレイヴにコンテキストスイッチする際に，システムメモリに対するアクセス権限を無効にし，エンクレイヴに割り当てた物理メモリに対するアクセス権限のみを有効にする．逆に，エンクレイヴからシステムにコンテキストスイッチする際は，システムメモリに対するアクセス権限を有効にし，エンクレイヴに割り当てたメモリに対するアクセス権限を無効にする．Keyspector では，システムからエンクレイヴにコンテキストスイッチする際にシステムメモリに対する読み込み権限も有効にすることにより，エンクレイヴがシステムメモリを参照できるようにする (図 3)．ただし，現在のところ，読み込み権限のみを有効にすることはできておらず，すべての権限を有効にしている．

IDS からシステムメモリにアクセスできるようにするために，ランタイムが IDS のメモリアドレス空間へのマッピングを行う．エンクレイヴの起動時にはまず，ローダが実行され，ランタイムがロードされる．この時に，ローダがエンクレイヴに割り当てられた物理メモリをランタイムのカーネルアドレス空間にマッピングする．ランタイム上で動作するアプリケーションは 1 つだけであるため，Keyspector のローダは同時にシステムメモリをユーザアドレス空間にマッピングする．その際に，U モードでのアクセスを許可し，読み込み権限のみを付与する．その後で，ランタイムが IDS をロードするため，IDS がスタックやヒープとして使用する仮想アドレスの範囲と重複しないようにする．IDS 以外のエンクレイヴ・アプリケーションはシステムメモリにアクセスする必要がないため，今後，IDS がシステムコールを実行してシステムメモリのマッピングを行えるようにする予定である．

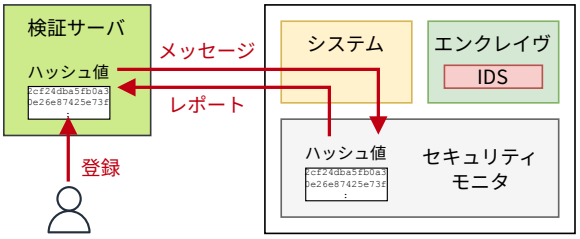


図 4 リモートアテステーションを用いたシステムメモリの共有制御

4.2 メモリ共有の制限

Keyspector はエンクレイヴ内で動作するソフトウェアのハッシュ値を用いて，システムメモリの共有を許可するかどうかを判定する．セキュリティモニタはエンクレイヴの起動時に，エンクレイヴ内で実行されるローダ，ランタイム，アプリケーションのハッシュ値の計算を行う．事前に登録しておいた正規の IDS のハッシュ値と比較を行い，一致した場合，システムメモリの共有を許可するフラグを有効にする．システムからエンクレイヴにコンテキストスイッチする際に，そのフラグが有効である場合はシステムメモリを共有し，有効でない場合は共有しないようにする．

リモートアテステーションと組み合わせる場合には，検証サーバでハッシュ値が一致するかどうかを判定する．まず，図 4 のようにセキュリティモニタがエンクレイヴ内で動作するソフトウェアのハッシュ値を計算し，署名する．そのレポートをシステム経由でリモートの検証サーバに送信し，検証を行う．リモートアテステーションではハッシュ値が正しければ検証に成功するが，さらに，事前に登録しておいた IDS のハッシュ値と一致するかどうかの判定も行う．一致する場合は，システムメモリの共有を許可するというメッセージに署名し，システム経由でセキュリティモニタに送信する．セキュリティモニタは受信したメッセージの検証を行い，結果に応じてシステムメモリの共有を許可するフラグを有効にする．この手法の利点は，IDS を更新した際にセキュリティモニタを変更せずに新しいハッシュ値を登録できることである．この手法は現在のところ，未実装である．

4.3 アドレス変換

エンクレイヴ内で動作する IDS は監視対象システムのページテーブルを用いて OS データの仮想アドレスを物理アドレスに変換する．そのために，IDS はシステムメモリ上にあるページテーブルの物理アドレスを取得する．このアドレスは satp レジスタに格納されているが，エンクレイヴの実行中はエンクレイヴ用のページテーブルのアドレスが格納されている．システムの実行中に使われていた satp レジスタの値は，システムからエンクレイヴにコンテキストスイッチする際にセキュリティモニタに保存されている．そこで，IDS は図 5 のように，システムコールを用いてランタイムを呼び出し，ランタイムは Supervisor Binary

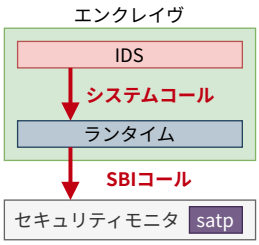
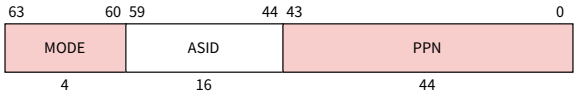
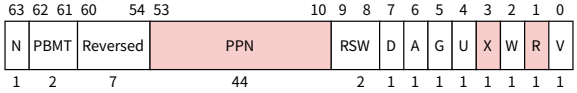


図 5 satp レジスタ値の取得



(a) satp レジスタ



(b) PTE

図 6 ページテーブルの内部構成

Interface (SBI) コールを用いてセキュリティモニタを呼び出す．セキュリティモニタは保存されている satp レジスタの値をランタイム経由で IDS に返す．

satp レジスタの値からページテーブルを特定した後，IDS はページテーブルをたどって仮想アドレスに対応するページテーブルエントリ (PTE) を見つける．RISC-V はいくつかのアドレス変換モードをサポートしており，用いられているモードは satp レジスタの MODE フィールドに格納されている (図 6(a))．現在の実装では，57 ビットの仮想アドレスをサポートし，5 レベルのページテーブルを用いる Sv57 に対応している．ページテーブルをたどっていき，R フィールドも X フィールドも 1 でない PTE に到達した場合は，PPN フィールドが指す次のレベルの PTE を調べる (図 6(b))．これらのどちらかが 1 となっている PTE に到達した場合は，PPN フィールドと仮想アドレスの下位 12 ビットのオフセットから物理アドレスを計算する．5 レベルをたどる前にこのような PTE に到達した場合は，ページサイズが 256TB，512GB，1GB，2MB のスーパーページとして扱う．アドレス変換の結果はキャッシュしておき，同じページの仮想アドレスについてはページテーブルをたどらない．

4.4 開発した IDS

Keyspector は LLView フレームワーク [9] を用いることで，エンクレイヴで動作する IDS が必要とする OS データのアドレス変換が透過的に行われるようにする．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ことを可能にするフレームワークである．具体的には，カーネルのヘッダファイルに含まれるグローバル変数，インライン関数，マクロなどを

```
for_each_online_cpu(i) {
    struct kernel_cpustat kcpustat;
    kcpustat_cpu_fetch(&kcpustat, i);
    idle = get_idle_time(&kcpustat, i);
    :
}
```

図 7 CPU 時間を取得するプログラム

使うことができ，カーネルのコードを再利用することができる．LLView は作成したプログラムをコンパイルして生成された LLVM の中間表現に対して，`load` 命令の前にアドレス変換の処理を埋め込む．RISC-V 用のクロスコンパイルに対応するために，LLView が用いる `clang` と `llc` に RISC-V 64 ビットアーキテクチャを指定し，`llc` には ABI として `lp64d` を指定した．

この LLView を用いて，システムメモリ上の OS データを解析して 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 を開発した．proc ファイルシステムは Linux カーネルの状態が格納された疑似ファイルを作成し，プロセス，ハードウェアおよびシステムリソースに関する情報を提供する．開発した IDS が作成する疑似ファイルを表 1 に示す．図 7 は CPU ごとの統計情報を取得するプログラムの一部である．このプログラムで使われている `for_each_online_cpu` は Linux カーネルのヘッダに定義されているマクロであり，`kernel_cpustat` 構造体を用いて情報を取得している．

5. 実験

Keyspector 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 Linux の proc ファイルシステムが提供するシステム情報のうち，表 1 の 10 種類の疑似ファイルを作成するために必要なシステム情報を収集する．比較として，監視対象システム内で proc ファイルシステムを読み出す従来手法を用いた．この実験では，仮想化ソフトウェアの QEMU 6.2.0 を用いて，RISC-V プロセッサのエミュレータを動作させた．エミュレーション環境には仮想 CPU を 4 コア，メモリを 2GB 割り当て，Linux 6.1.32 を動作させた．実験に用いたマシンの CPU は Intel Core i7-14700，メモリは 64GB であった．

5.1 IDS の動作確認

Keyspector を用いてエンクレイヴ内で IDS を実行し，監視対象システムのメモリ上の OS データを収集して作成した疑似ファイルの内容を表示させた．図 8 に出力されたシステム情報の一部を示す．この出力結果と監視対象システム内でエンクレイヴの実行直後に proc ファイルシステムを読み出した時の出力を比較したところ，出力が一致することを確認した．この結果から，エンクレイヴ内の IDS が

表 1　開発した IDS が作成する proc ファイルシステムの疑似ファイル

ファイルパス	提供されるシステム情報
/proc/meminfo	システムのメモリ使用率
/proc/net/tcp	TCP ソケットのコネクション情報
/proc/stat	システムの統計情報 (各種状態で消費された CPU 時間の合計値など)
/proc/sys/kernel/osrelease	Linux カーネルのバージョン
/proc/sys/kernel/pid_max	プロセス ID の最大値
/proc/tty/drivers	カーネルにロードされている tty デバイス
/proc/uptime	システム起動時から経過した時間とアイドル時間
/proc/<PID>/auxv	実行時にプロセスに渡された Executable and Linkable Format (ELF) インタープリタの情報
/proc/<PID>/stat	プロセスの状態についての情報
/proc/<PID>/status	/proc/<PID>/stat と/proc/<PID>/statm の情報をわかりやすく整形したもの

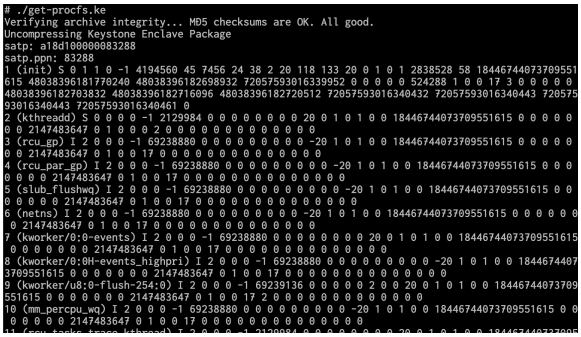


図 8　IDS の実行結果

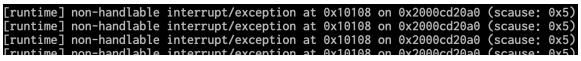


図 9　不正な IDS の実行結果

システム情報を正しく取得できることが確認できた．

次に，IDS のプログラムを少し変更して，ハッシュ値がセキュリティモニタに登録したものと異なるようにしてエンクレイヴ内で実行した．この IDS がシステムメモリにアクセスした時のランタイムの出力を図 9 に示す．この結果より，侵入者がエンクレイヴ内で不正なプログラムを実行してもシステムメモリ上の情報は盗聴できないことを確認した．

5.2　取得時間の測定

Keyspector を用いてエンクレイヴ内で IDS を実行し，システム情報を収集するのにかかる時間を測定した．監視対象システム内で動作していたプロセス (カーネルスレッドを含む) は IDS を実行するためのものを含めて 63 個であった．システム情報の取得時間を 10 回測定した平均値と標準偏差を図 10 に示す．Keyspector は監視対象システム内でシステム情報を取得する従来手法と比べて，取得時間が 2.4 倍に増加した．この原因としては，アドレス変換をソフトウェアで行うことによるオーバヘッドが考えられる．

さらに詳細に Keyspector によるオーバヘッドを調べるために，IDS がそれぞれの疑似ファイルを作成するのに必要なシステム情報を取得するのにかかる時間を個別に調べた．10 回測定した平均値と標準偏差を図 11 に示す．プロセス毎の情報に関してはすべてのプロセスの情報を取得するのにかかった時間の合計となっている．また，それぞれのシステム情報を取得するために IDS が行ったアドレス変換の回数を図 12 に示す．

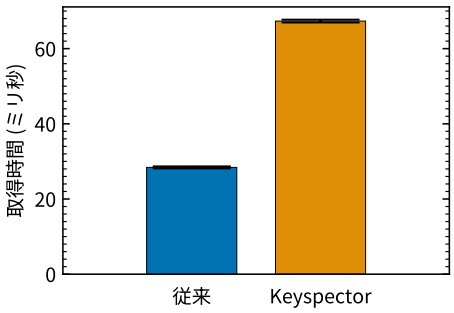


図 10　システム情報の取得時間

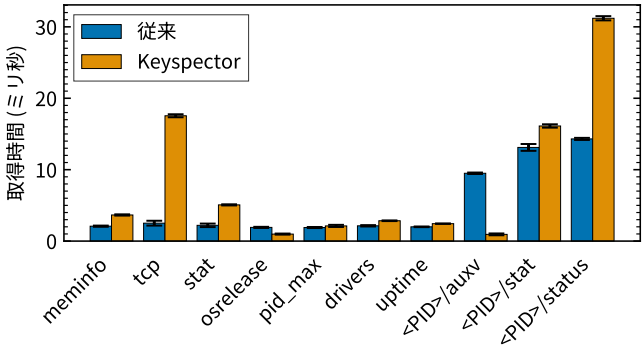


図 11　個別のシステム情報の取得時間

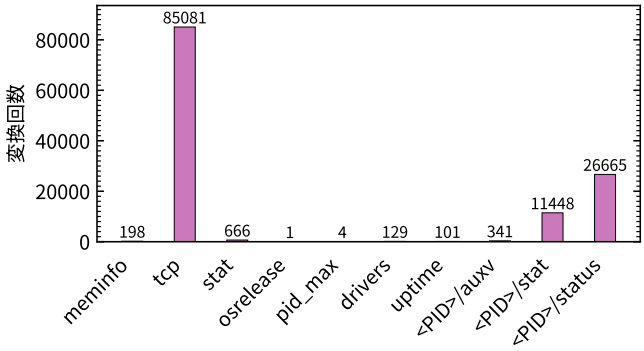


図 12　IDS が行ったアドレス変換回数

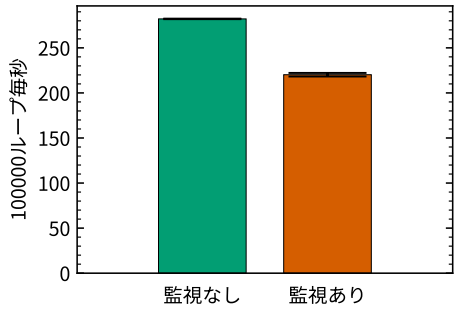


図 13 Dhrystone の実行結果

ほとんどの項目において従来手法に比べて Keyspector での取得時間の方が長くなっている．tcp やプロセスの status の取得時間が他の項目に比べて大幅に長いのは，アドレス変換が非常に多く行われたためだと考えられる．一方，osrelease やプロセスの auxv の取得時間は従来手法よりも短くなっている．これは，取得する情報が少なく，監視対象システム内で proc ファイルシステムを読み出すためのシステムコールを実行するオーバヘッドがエンクレイヴ内でアドレス変換を行うオーバヘッドよりも大きくなったためだと考えられる．

5.3 CPU 性能への影響

監視対象システム内で Dhrystone ベンチマークを実行し，エンクレイヴ内で IDS を動作させていない時と動作させている時に CPU 性能の測定を行った．この実験では，IDS は 10 ミリ秒間隔でシステム情報の取得を行うようにした．ベンチマーク結果を図 13 に示す．この結果より，IDS の実行によってシステムの CPU 性能が 22%低下することが分かった．このオーバヘッドは IDS によるシステム情報の取得頻度を減らせば小さくなると考えられる．

6. 関連研究

SSdetector [6] は Intel SGX と CPU の動作モードでの一つであるシステムマネジメントモード (SMM) を組み合わせることで，安全な IDS 実行を可能にするシステムである．IDS を SGX のエンクレイヴ内で実行することで，IDS の盗聴や改ざんを防ぐことができる．一方で，SGX のエンクレイヴからはシステムメモリに直接アクセスすることができないため，SMM で動作する BIOS 内のプログラムを呼び出してメモリデータを取得する．そのために，OCALL を用いて一旦，エンクレイヴ外のコードを呼び出し，システムマネジメント割り込み (SMI) を発生させる必要がある．また，エンクレイヴと BIOS 内のプログラム以外による盗聴や改ざんを防ぐために，やりとりされるデータは暗号化され，整合性検査が行われる．これらのオーバヘッドが大きいため，監視性能が低下する．加えて，SMM でのプログラムの実行中はシステム全体が停止するため，システム性能も低下する．また，エンクレイヴを動作させているプロセスが停止させられると IDS も停止するため，IDS に定期的にハートビートを送って動作を確認する必要がある．

SGX を用いたネットワークベース IDS として S-NFV [11] や SEC-IDS [12] がある．S-NFV はオープンソースの IDS である Snort の内部状態をエンクレイヴ内に格納する．エンクレイヴ内に置いた内部状態は外部からの攻撃を受けない安全な API 経由で利用することで，ネットワークフローごとの情報などの盗聴を防ぐ．SEC-IDS は Snort をほとんど修正なしにエンクレイヴ内で実行する．エンクレイヴ内で既存の IDS を動作させるために，Graphene-SGX ライブラリ OS [13] を用いる．Snort はエンクレイヴ内で安全に実行できるが，Snort が取得するまでの間にパケットを書き換えられると，攻撃を正しく検知できない可能性がある．

ITZ ライブラリ [7] は Arm TrustZone を用いて仮想マシン (VM) イントロスペクション (VMI) を可能にするライブラリである．監視対象システムが動作しているノーマルワールドをある種の VM とみなして，セキュアワールドからノーマルワールドのメモリを LibVMI [14] に似た API を用いて監視することができる．セキュアワールドではマイクロカーネルと仮想マシンモニタ (VMM) が動作し，マイクロカーネル上で ITZ ライブラリを用いた VMI ツールが動作する．ノーマルワールドからセキュアワールドにアクセスすることはできないため，監視対象システムへの侵入者によって VMI ツールが無効化されることはない．一方で，セキュアワールドは非常に高い権限を持つため，VMI ツールはシステムの監視以上のことができてしまう可能性がある．

Trusted Monitor [15] は TrustZone のセキュアワールド内で CPU の性能カウンタを用いて異常検知を行う IDS を安全に実行することができる．この IDS は性能カウンタの値を収集し，正常にシステムが動作している時の性能カウンタの値を用いて学習させた機械学習モデルを用いて検知を行う．ノーマルワールドで動作するエージェントが IDS を定期的に呼び出し，一定時間以上呼び出されない場合はウォッチドッグタイマによってシステムがリセットされる．ノーマルワールドから性能カウンタに干渉できないように設定するため，ノーマルワールドでは性能カウンタを利用することができない．

Elasticlave [16] はエンクレイヴ間およびエンクレイヴとホスト間でメモリの柔軟な共有を可能にするメモリモデルである．Keystone にはエンクレイヴ間でメモリを共有する機能はないため，エンクレイヴとホスト間の共有メモリを用いてホスト経由でデータを交換する必要がある．Elasticlave ではエンクレイヴ間でメモリを直接共有することができ，エンクレイヴごとに異なるアクセス権を設定したり，エンクレイヴがそのアクセス権を変更したりすることができる．さらに，共有メモリに排他的にアクセスする

こともできる．これにより，セキュリティと柔軟性のバランスをとることができる．

TVMmonitor [17] は RISC-V Confidential VM 拡張 (CoVE) [18] で保護された VM (TVM) への侵入を検知する監視システムである．TVM 同士の隔離は TEE セキュリティマネージャ (TSM) によって行われる．監視システムを TVM 内で動作させることで，TSM によって監視対象の TVM から隔離して安全に実行することができる．監視用 TVM は監視対象 TVM との間の共有メモリを用いて情報を取得することで，ネットワーク通信を用いるより高速に監視を行うことができる．しかし，TVMmonitor はクラウドで利用される Confidential VM を想定したシステムとなっているため，IoT 機器での利用には適していない．

7. まとめ

本稿では，RISC-V プロセッサの TEE である Keystone を用いて IDS を安全に実行し，監視対象システムのメモリデータを効率よく取得することを可能にする Keyspctor を提案した．Keyspector はセキュリティモニタにおいてアクセス権限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が監視対象システムのメモリに直接アクセスすることを可能にする．実験結果から，Keyspector を用いて監視対象システムの proc ファイルシステムが提供するシステム情報を取得できることを確認した．

今後の課題は，実機を用いて Keyspector の性能を測定することである．また，システムメモリの共有制限をリモートアテステーションと組み合わせることができるように実装を行う．proc ファイルシステムが提供する他のシステム情報も取得できるようにし，取得した情報を用いてどのように侵入を検知するかについても検討する．

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] 総務省 情報流通行政局 情報通信政策課 情報通信経済室 統計企画係：令和 6 年版 情報通信白書 データ集，総務省（オンライン），入手先 ⟨https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r06/html/datashu.html⟩（参照 2025-04-17）．

[2] 国立研究開発法人情報通信研究機構サイバーセキュリティネクサス：NICTER 観測レポート 2024，国立研究開発法人情報通信研究機構（オンライン），入手先 ⟨https://csl.nict.go.jp/report/NICTER_report_2024.pdf⟩（参照 2025-04-17）．

[3] Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K. and Zhou, Y.: Understanding the Mirai Botnet, *26th USENIX Security Symposium*, pp. 1093–1110 (2017).

[4] Molina, J.: Learn how to control every room at a luxury hotel remotely: The dangers of insecure home automation deployment, *Black Hat USA*, Vol. 2014 (2014).

[5] Miller, C. and Valasek, C.: Remote exploitation of an unaltered passenger vehicle, *Black Hat USA*, Vol. 2015, No. S 91, pp. 1–91 (2015).

[6] 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).

[7] 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).

[8] Lee, D., Kohlbrenner, D., Shinde, S., Asanovic, K. and Song, D.: Keystone: An Open Framework for Architecting Trusted Execution Environments, *Proceedings of the Fifteenth European Conference on Computer Systems* (2020).

[9] 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).

[10] Cerdeira, D., Martins, J., Santos, N. and Pinto, S.: ReZone: Disarming TrustZone with TEE Privilege Reduction, *31st USENIX Security Symposium*, pp. 2261–2279 (2022).

[11] 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).

[12] Kuvaiskii, D., Chakrabarti, S. and Vij, M.: Snort Intrusion Detection System with Intel Software Guard Extension (Intel SGX), *arXiv:1802.00508* (2018).

[13] Tsai, 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).

[14] Payne, B. D.: Simplifying Virtual Machine Introspection Using LibVMI, Technical report, Sandia National Laboratories (2012).

[15] Eichler, C., Röckl, J., Jung, B., Schlenk, R., Müller, T. and Hönig, T.: Profiling with trust: system monitoring from trusted execution environments, *Des. Autom. Embedded Syst.*, Vol. 28, No. 1, pp. 23–44 (2024).

[16] Yu, J. Z., Shinde, S., Carlson, T. E. and Saxena, P.: Elasticlave: An Efficient Memory Model for Enclaves, *31st USENIX Security Symposium*, pp. 4111–4128 (2022).

[17] 梶原悠大，光来健一：RISC-V CoVE における TEE VM の柔軟で効率のよい監視システム，*SWoPP 2024* (2024).

[18] Sahita, R., Shanbhogue, V., Bresticker, A., Khare, A., Patra, A., Ortiz, S., Reid, D. and Kanwal, R.: CoVE: Towards Confidential Computing on RISC-V Platforms, *Proceedings of the 20th ACM International Conference on Computing Frontiers*, pp. 315–321 (2023).