Vol.2020-OS-149 No.3
2020/5/28

SGX向け実行環境SCONEを用いたVMの安全な監視機構

河村 拓実[1]　光来 健一[1]

概要：IaaS 型クラウドが提供している仮想マシン（VM）はインターネット経由で攻撃を受けやすいため，侵入検知システム（IDS）を用いて監視を行う必要がある．IDS を安全に実行するための手法として IDS を VM の外で実行する IDS オフロードが提案されているが，まだ外部の攻撃者やオフロードした IDS を運用するクラウドの管理者から攻撃を受ける可能性がある．この問題を解決するために，CPU のセキュリティ機構である Intel SGX を用いてエンクレイヴと呼ばれる保護領域内に IDS をオフロードするシステムが提案されてきた．しかし，VM 内の情報を取得するにはカーネルレベルのプログラミングが必要となり，一般の開発者には難しい．そこで本稿では，SGX 向け実行環境である SCONE を利用して OS の標準インタフェースを提供することで，既存の IDS をエンクレイヴ内にオフロード可能にするシステム SCwatcher を提案する．SCwatcher は標準 C ライブラリのインタフェースに加えて，監視対象 VM のシステム情報を取得可能な proc ファイルシステムをエンクレイヴ内の IDS に対して提供する．この proc ファイルシステムは SCONE が提供する非同期システムコール機構を利用して監視対象 VM の OS データを透過的に取得する．さらに，IDS のコンパイル時に標準ファイル関数の呼び出しを置換することにより，エンクレイヴ内の proc ファイルシステムにアクセスさせる．SGX 仮想化をサポートした Xen-SGX 4.7 に SCwatcher を実装し，既存の netstat コマンドの動作確認と性能測定を行った．

1. はじめに

近年，仮想マシン（VM）を提供する IaaS 型クラウドが普及している．ユーザは VM に好きな OS やアプリケーションをインストールし，自由にシステムを構築することができる．一方で，クラウド内の VM はインターネット経由で攻撃を受けやすいため，侵入検知システム（IDS）を用いて VM を監視する必要がある．攻撃者が VM に侵入した際に IDS が無効化されるのを防ぐために，IDS を VM の外で実行する IDS オフロード [1] と呼ばれる手法が用いられている．この手法により，VM に侵入した攻撃者は IDS を攻撃できなくなるが，クラウド外部の攻撃者やクラウドの内部犯から IDS が攻撃を受ける恐れがあり，オフロードした IDS の安全性はまだ十分に確保できていない．

この問題を解決するために，CPU のセキュリティ機構である Intel SGX を用いて IDS を安全にオフロードするシステム [2] が提案されてきた．このシステムでは，エンクレイヴと呼ばれる保護領域内で IDS を実行することにより，IDS の改竄や VM から取得した情報の漏洩を防ぐことができる．エンクレイヴ内の IDS は監視対象 VM のメモリを解析して OS データを取得することによりシステムの監視を行う．しかし，監視に必要な情報を取得するにはカーネルレベルのプログラミングを行わなければならず，一般の開発者にとっては難しい．また，エンクレイヴ内では SGX 向けの専用ライブラリを用いる必要があり，OS が標準的に提供するインタフェースを利用することができない．

本稿では，SGX 向け実行環境である SCONE [3] を利用して OS の標準インタフェースを提供することで，既存の IDS をエンクレイヴ内にオフロードして実行可能にするシステム SCwatcher を提案する．SCONE はエンクレイヴ内で既存のアプリケーションを実行するための実行環境である．SCwatcher は SCONE が提供する標準 C ライブラリのインタフェースを用いて既存の IDS をコンパイル・実行する．SCONE はこのインタフェースを通してエンクレイヴが動作している OS の情報しか提供しないため，SCwatcher はさらに監視対象 VM のシステム情報を取得可能な proc ファイルシステムを提供する．この proc ファイルシステムは SCONE が提供する非同期システムコール機構を利用して，監視対象 VM の OS データを透過的に取得する．

我々は SGX 仮想化をサポートした Xen-SGX 4.7 [4] と SCONE を用いて SCwatcher を実装した．SCwatcher は IDS を実行するための専用の VM を作成し，その中でエンクレイヴを用いて IDS を実行する．エンクレイヴ内で動作する proc ファイルシステムの開発には LLView [2] を用いた．LLView を用いてプログラム変換を行うことにより，

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

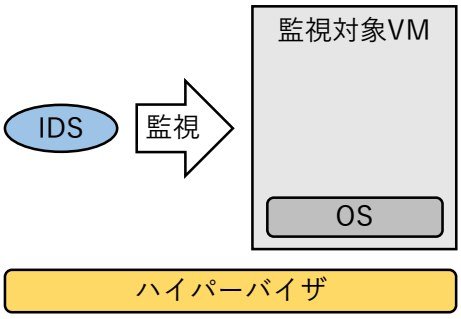

図 1 IDS オフロード

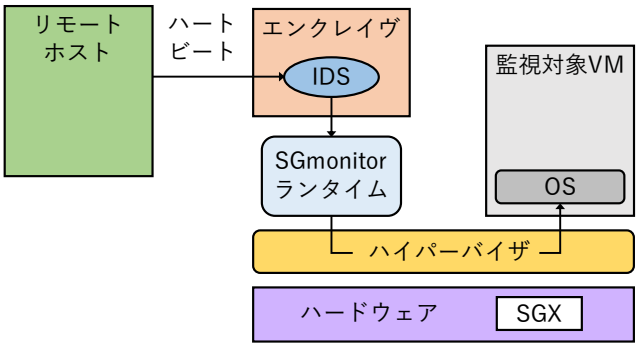

図 2 SGmonitor のシステム構成

Linux のソースコードをほぼそのまま利用して proc ファイルシステムを開発し，監視対象 VM の OS データを透過的に取得させることができる．proc ファイルシステムは取得した OS データを基に疑似ファイルを作成して IDS に提供する．IDS がエンクレイヴ内 proc ファイルシステムにアクセスできるようにするために，IDS のコンパイル時に標準ファイル関数の呼び出しを置換する．SCwatcher を用いてエンクレイヴ内で既存の netstat コマンドを実行し，監視対象 VM のネットワーク接続状況が取得できることを確認した．また，netstat の実行時間を従来の IDS オフロードと比較し，SCwatcher のオーバヘッドについて調べた．

以下，2 章ではクラウドにおける IDS オフロードが抱える問題点および，SGX を用いた IDS の保護について述べる．3 章では既存の IDS を安全にオフロードするシステム SCwatcher を提案し，4 章で SCwatcher の実装について述べる．5 章では SCwatcher の性能を調べるために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. クラウドにおける IDS オフロード

IDS オフロード [1] は，**図 1** のように監視対象 VM の外で IDS を動かす手法である．IDS オフロードを用いることにより，VM 内に侵入されたとしても IDS は VM の外にあるため，侵入者に無効化される恐れはない．監視対象 VM 内で IDS を動かす従来の手法と異なり，オフロードした IDS は監視対象 VM のメモリを解析し，OS が管理しているデータを取得する．これにより，監視対象 VM の OS から直接情報を取得する従来の手法と同じように VM への攻撃を検知することが可能となる．例えば，監視対象 VM 内のネットワーク接続状況を調べることで不正な通信を検知することができる．

しかし，IDS オフロードを用いてもまだ IDS が攻撃を受ける可能性がある．クラウド外部の攻撃者からオフロードした IDS が攻撃を受ける恐れがあるためである．また，オフロードした IDS を運用するクラウド内に内部犯がいる可能性もある．実際に，Google の管理者がユーザーの個人情報を盗み見てプライバシを侵害するという事件が発生している [6]．また，サイバー犯罪の 28%が内部犯行であるという調査結果 [7] や，管理者の 35%が機密情報を盗み見たことがあるという調査結果 [8] もある．

この問題を解決するために，オフロードした IDS を SGX を用いて保護し，VM を安全に監視するシステム SGmonitor [2] が提案されている．SGX は Intel 製 CPU が提供する新しいセキュリティ機構である．SGX を利用することでメインメモリ上にエンクレイヴと呼ばれる保護領域を作成し，その中でアプリケーションを安全に実行することができる．エンクレイヴ内のプログラムは実行開始時に電子署名が検査されるため，改竄されたプログラムは実行することができない．また，エンクレイヴのメモリは整合性が保たれるため，実行中のプログラムの改竄も不可能である．さらに，エンクレイヴのメモリは暗号化されるため，データの漏洩を防ぐことができる．

SGmonitor では**図 2** のように，オフロードした IDS をエンクレイヴ内で実行することで IDS を攻撃から守る．エンクレイヴ内の IDS は監視対象 VM の OS データを取得するために，まず OCALL を用いてエンクレイヴ外の SGmonitor ランタイムを呼び出す．OCALL はエンクレイヴ内から外部の信頼できないコードを呼び出すための機構である．SGmonitor ランタイムはハイパーコールを用いてハイパーバイザを呼び出し，監視対象 VM のメモリから OS データを取得する．SGX はエンクレイヴごと IDS を停止されるのを防ぐことができないため，クラウド外部の信頼できるリモートホストから IDS にハートビートを送ることで，IDS が正常に動作していることを確認する．

しかし，SGmonitor を用いてエンクレイヴ内で動作する IDS を開発するには，OS カーネルのデータ構造を用いてカーネルレベルのプログラミングを行う必要がある．オフロードした IDS は標準的な OS インタフェースを用いて OS の情報を取得することができないためである．IDS は OS の内部構造に関する知識を用いて監視対象 VM のメモリを解析しなければならない．このような IDS の開発はカーネル内で動作する IDS 以外では行われておらず，一般の IDS の開発者にとっては難しい．その上，監視対象 VM 内の OS にも大きな影響を受けるため，OS のバージョン

ごとに IDS を開発する必要がある．また，エンクレイヴ内の IDS は SGX 向けに提供されている SDK [9] を用いて開発する必要があり，標準的に用いられているライブラリを利用することもできない．

3. SCwatcher

3.1 脅威モデル

本稿では，SGmonitor と同様に以下のような脅威モデルを考える．まず，クラウドプロバイダは信用できるものとする．クラウドプロバイダにとって，ユーザからの信用を失うことは致命的であるため，この仮定は様々な研究で広く用いられている [10], [11], [12], [13], [14], [15]．プロバイダが提供するクラウド内のハードウェアも信頼する．SGX の脆弱性は考えず，エンクレイヴ内のセキュリティは担保されるものとする．また，クラウド内のハイパーバイザは様々な既存手法を用いることで正常に動作していると仮定する．例えば，TPM を使用したリモートアテステーションにより，クラウドプロバイダやユーザはハイパーバイザが正常に起動したことを確認することができる．また，システム管理モードなどのハードウェア機構を用いることにより，実行中のハイパーバイザの改竄を検知することもできる [16], [17], [18], [19]．一方で，オフロードした IDS を実行するシステムのうち，エンクレイヴ以外の部分は信用しない．本稿では，外部の攻撃者や信頼できないクラウド内の管理者が IDS を攻撃する状況を想定する．

3.2 SGX を用いた既存 IDS のオフロード

本稿では，SGX 向け実行環境である SCONE [3] を利用して標準的な OS インタフェースを提供することにより，エンクレイヴ内で既存の IDS を実行可能にするシステム SCwatcher を提案する．SCONE は既存のアプリケーションをエンクレイヴ内で安全に実行するための実行環境である．SCONE のライブラリがエンクレイヴ内のアプリケーションに標準 C ライブラリのインタフェースを提供し，システムコールを発行した時にはエンクレイヴ外の OS を呼び出す．その際に，非同期システムコール機構を用いることによりオーバヘッドを削減する．SCONE が提供するコンパイラを用いることで，既存のアプリケーションのソースコードからエンクレイヴ内で動作する実行ファイルを生成する．

SCwatcher のシステム構成を**図 3** に示す．SCwatcher は，エンクレイヴ内で SCONE ライブラリを用いて IDS を動かす．しかし，SCONE ライブラリを通して取得できるのはエンクレイヴが動作している OS の情報であるため，そのままではオフロードした IDS が監視対象 VM 内の情報を得ることはできない．そこで，SCwatcher では IDS の多くがプロセスやネットワークなどのシステム情報を取得するために利用している proc ファイルシステムを提供する．

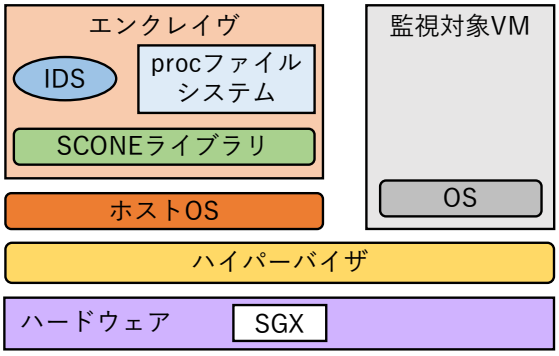

図 3 SCwatcher のシステム構成

この proc ファイルシステムは IDS が監視対象 VM 内のシステム情報を標準的な OS インタフェースを用いて取得することを可能にする．これにより，IDS は監視対象 VM の OS のバージョンの違いによる影響を受けない．エンクレイヴ内 proc ファイルシステムは監視対象 VM のメモリ上の OS データを取得し，その情報を基に疑似ファイルを作成する．この疑似ファイルはディスク上に格納される通常のファイルとは異なり，アクセスしたときに動的に内容が作成される特殊なファイルである．

エンクレイヴ内 proc ファイルシステムは OCALL を用いてエンクレイヴ外のハイパーバイザを呼び出すことで，監視対象 VM のメモリにアクセスする必要がある．しかし，ソースコードが公開されていない SCONE にこのような機能を追加するのは難しい．そこで SCwatcher では，エンクレイヴ外のホスト OS 内に VM メモリデバイスを用意し，SCONE の非同期システムコール機構を用いてこのデバイスにアクセスしてハイパーコールを発行する．VM メモリデバイスは信用できないホスト OS 内で動くため，SCwatcher はエンクレイヴとハイパーバイザの間でデータを暗号化し，整合性検査を行う．

SCwatcher の proc ファイルシステムは IDS のコンパイル時に IDS 本体とリンクされる．しかし，そのままでは IDS が proc ファイルシステムにアクセスしようとすると，SCONE ライブラリによってエンクレイヴが動いているホスト OS が呼び出されてしまう．そこで，IDS のコンパイル時に fopen などの標準ファイル関数の呼び出しを置換することで，必要に応じてエンクレイヴ内の proc ファイルシステムにリダイレクトする．ホスト OS は共有ライブラリを用いて関数を置き換える LD_PRELOAD 機構を提供しているが，エンクレイヴ内ではその機構を用いることはできない．標準ファイル関数の代わりに呼び出される疑似ファイル管理機構の関数は，proc ファイルシステムの疑似ファイルに対して標準ファイル関数の処理をエミュレートする．例えば，IDS が疑似ファイルをオープンする時に疑似ファイルの中身を作成する．疑似ファイルを読み込む際には，オープン時に作成した疑似ファイルの内容を返す．

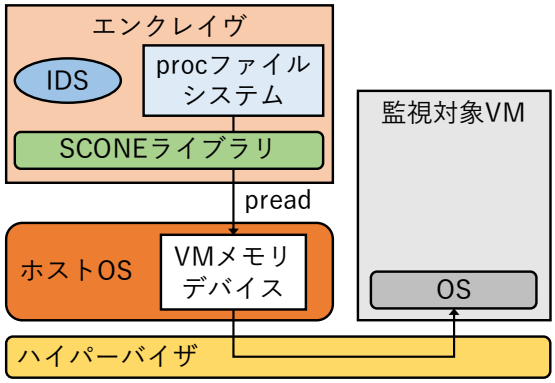

図 4 OS データの取得

4. 実装

我々は SGX 仮想化をサポートした Xen-SGX 4.7 [4] と SCONE を用いて SCwatcher を実装した．

4.1 VM メモリデバイス

SCONE は，エンクレイヴ内でシステムコールを発行する必要がある時にはエンクレイヴ外のホスト OS を呼び出して実行させることができる．その際に，OCALL を用いず，共有バッファ経由で引数と返り値を受け渡す．SCwatcher の proc ファイルシステムが監視対象 VM の OS データを取得する際には，この非同期システムコール機構を利用して**図 4** のように，ホスト OS 内に用意した VM メモリデバイスにアクセスする．このデバイスアクセスにはファイルオフセットを指定してデータの読み込みを行う pread システムコールを用いる．

pread の引数として指定するファイルオフセットには，アクセスする OS データの仮想アドレスを対応させる．しかし，カーネルの仮想アドレスの値はファイルオフセットとしては大きすぎて不正な値となるため，12 ビット右シフトした値を pread の引数に指定する．この値は VM メモリデバイス内で 12 ビット左シフトして仮想アドレスに戻す．この時，仮想アドレスのページ内オフセットを示す下位 12 ビットが失われてしまうが，SCwatcher はページ単位で仮想アドレスを指定するため問題にはならない．

pread システムコールで呼び出された VM メモリデバイスは，読み出し処理を行う関数でハイパーコールを実行することによりハイパーバイザを呼び出す．ハイパーバイザは VM の仮想 CPU から CR3 レジスタの値を取得して VM のメモリ上にあるページテーブルを参照し，取得しようとしているデータの仮想アドレスを物理アドレスに変換する．そして，この物理アドレスに対応する VM のメモリページをコピーし，ページ単位で OS データを取得する．監視対象となる VM のドメイン ID は，VM メモリデバイスのマイナー番号で指定する．

取得した OS データはエンクレイヴに返す前にハイパーバイザ内で暗号化し，エンクレイヴ内 proc ファイルシステムで復号する．これにより，信頼できない VM メモリデバイスでの情報漏洩を防ぐことができる．データの暗号化と復号化を行うために，ハイパーバイザと proc ファイルシステムにそれぞれ wolfSSL [20] の AES 関数を移植した．同様に，エンクレイヴからハイパーバイザに pread システムコールで渡す仮想アドレスも暗号化する必要があるが，これについては現在，未実装である．また，改竄を検出するためにハッシュ値を用いて整合性検査を行う必要もある．

4.2 エンクレイヴ内 proc ファイルシステム

Linux のソースコードをできるだけ用いてエンクレイヴ内 proc ファイルシステムの開発を行うために LLView [2] を用いた．LLView は，Linux のヘッダファイルをインクルードしてカーネル構造体，マクロ，インライン関数を使ったプログラミングを可能にする．プログラムをコンパイルする際には，生成される LLVM の中間表現のプログラム変換を行う．それにより，proc ファイルシステムが OS データにアクセスしようとした時に，監視対象 VM の OS データを取得させる．

具体的には，中間表現の load 命令の直前に OS データを取得するための関数呼び出しを挿入する．この関数は pread システムコールを用いて非同期システムコール機構経由で VM メモリデバイスにアクセスし，取得した OS データを復号してエンクレイヴ内のメモリに格納する．そして，そのメモリ上のデータを読み込むように load 命令の変換を行う．取得した OS データはハッシュ表を用いてキャッシュする．これにより，proc ファイルシステムが同じ OS データを必要とした際には再度，VM メモリデバイスにアクセスする必要はない．

現在のところ，エンクレイヴ内 proc ファイルシステムは，/proc/net/tcp と/proc/net/udp の疑似ファイルを IDS に提供する．これらの疑似ファイルには，TCP 通信や UDP 通信についての通信先の IP アドレスやポート番号，ステートなどの情報を格納する．tcp 疑似ファイルは，LISTEN 状態のソケットが登録されたハッシュ表と ESTABLISHED などの状態のソケットが登録されたハッシュ表のエントリをすべてたどることで作成する．同様に，udp 疑似ファイルは UDP のソケットが登録されたハッシュ表のエントリをすべてたどることで作成する．疑似ファイルの作成に必要な情報は sock 構造体から取得する．

4.3 疑似ファイル管理機構

SCwatcher は IDS が標準ファイル関数を呼び出す際に SCONE ライブラリではなく，**図 5** のように疑似ファイル管理機構を呼び出させる．そのために，IDS のコンパイル時に LLVM の中間表現に対してプログラム変換を行う．

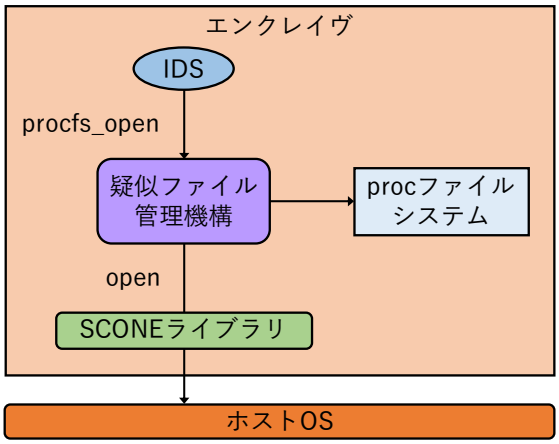

図 5 proc ファイルシステムへのアクセス

call 命令で呼び出される標準ファイル関数を procfs_というプレフィックスの付いた関数に置換することで，疑似ファイル管理機構が必要に応じてエンクレイヴ内 proc ファイルシステムにアクセスする．

例えば，IDS が open 関数を実行しようとした場合，procfs_open 関数が呼び出される．procfs_open 関数ではまず，引数のファイルパスのトップディレクトリが proc かどうか調べて，オープンしようとしているファイルが proc ファイルシステムの疑似ファイルか通常のファイルかを判別する．相対パスが指定されている場合にはカレントディレクトリを取得して判別する．通常ファイルの場合，標準ファイル関数の open 関数を実行し，その返り値を procfs_open 関数の呼び出し元に返す．

疑似ファイルであった場合，疑似的なオープン処理を行う．疑似ファイルをオープンする際には，procfs_open 関数は疑似ファイル番号を 0 から順番に割り当てる．この疑似ファイル番号に固定値（10000）を足したものを疑似ファイルディスクリプタとする．このように大きな値にするのは，通常ファイルをオープンした時に割り当てられるファイルディスクリプタの値と衝突するのを避けるためである．次に，疑似ファイルを管理するための PFILE 構造体を初期化し，proc ファイルシステムを呼び出して作成した疑似ファイルの中身のデータを格納する．そして，疑似ファイル番号をインデックスとする PFILE 構造体の配列に格納する．最後に，procfs_open 関数の返り値として疑似ファイルディスクリプタの値を返す．

procfs_fopen 関数の場合も同様の処理を行うが，疑似ファイルディスクリプタの代わりに疑似ファイルポインタを返す．疑似ファイルポインタは固定のベースアドレスに疑似ファイル番号を足したアドレスとする．これにより，FILE 構造体のポインタである通常のファイルポインタと区別することができる．また，procfs_opendir 関数の場合も同様に，通常の DIR 構造体のポインタと区別するために，疑似的なポインタを返す．

表 1 疑似ファイル管理機構の関数

関数名	処理
procfs_read/fread	疑似ファイルのデータを PFILE 構造体に格納されたファイルオフセットの位置から読み，ファイルオフセットを更新
procfs_fgets	疑似ファイルのデータをファイルオフセットの位置から一行分読み，ファイルオフセットを更新
procfs_fileno	疑似ファイルポインタからベースアドレスを引いて疑似ファイル番号を取得し，固定値を足して疑似ファイルディスクリプタにして返す
procfs_feof	ファイルオフセットがファイルサイズと等しくなったら 1 を返す
procfs_setvbuf	何もしない
procfs_close/fclose	PFILE 構造体を解放

ファイルディスクリプタを引数にとる関数が呼び出されると，その値から固定値（10000）を引いた値が疑似ファイル番号の範囲内であれば疑似的なファイル処理を行う．それ以外の場合には対応する標準ファイル関数を呼び出す．一方，ファイルポインタを引数にとる関数が呼び出されると，ファイルポインタの値からベースアドレスを引いた値が疑似ファイル番号の範囲内であれば疑似的なファイル処理を行う．DIR 構造体のポインタを引数にとる関数の場合も同様である．それぞれの関数での疑似的なファイル処理を**表 1** に示す．

5. 実験

SCwatcher を用いて既存のエンクレイヴ内で netstat コマンドを実行して動作確認を行った．netstat は proc ファイルシステムから取得した情報を基にネットワークの接続状況を表示するコマンドであり，IDS から呼び出して使われることが多い．SCwatcher の性能を調べるために，エンクレイヴを用いない従来の IDS オフロードと実行時間を比較した．実験に使用したマシンの CPU は Intel Core i7-8700，メモリは 16GB であった．仮想化システムには Xen SGX 4.7 を使用し，IDS を動作させる VM と監視対象 VM にはそれぞれ仮想 CPU を 2 個，メモリを 2GB 割り当てた．IDS を動作させる VM には SGX ドライバ 2.5.0 をインストールした．

5.1 疑似ファイルの内容の確認

エンクレイヴ内 proc ファイルシステムにアクセスし，/proc/net/udp の疑似ファイルの内容を表示するプログラムを SCwatcher を用いて実行した．監視対象 VM 内で疑似ファイルを表示した結果と比較し，監視対象 VM の UDP 通信に関する情報を正しく取得できていることを確認した．同様に，/proc/net/tcp にアクセスして内容を出力したところ，ソケットの参照カウンタなどの情報が正

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 192.168.0.84:22         192.168.0.82:60550      ESTABLISHED
tcp        0      0 192.168.0.84:22         192.168.0.83:52356      ESTABLISHED
```

図 6 netstat の実行結果

しく取得できていないことが分かった．原因は不明であるが，実際にこれらの情報を IDS が利用することは少ない．

5.2 netstat の実行結果

SCwatcher を用いてエンクレイヴ内で netstat を実行し，監視対象 VM 内で実行した場合と出力結果を比較した．実験の結果，**図 6** に示すように，SCwatcher でも監視対象 VM のネットワーク接続状況を取得することができた．この実行結果は監視対象 VM 内で得られる情報と同じであった．

次に，SCwatcher を用いた場合の netstat の実行時間を測定した．比較のために，エンクレイヴを用いない従来の IDS オフロードの場合についても測定を行った．この場合も，IDS は SCwatcher と同じ VM にオフロードし，SCwatcher の proc ファイルシステムを使って監視対象 VM の情報を取得した．**図 7** に示すように，SCwatcher における netstat の実行時間は従来手法の 5.0 倍となり，大幅に性能が低下することが分かった．

5.3 SCwatcher のオーバヘッドの原因

SCwatcher のオーバヘッドの原因を調べるために，proc ファイルシステムを用いずに監視対象 VM のプロセス一覧を取得するのにかかる時間を先行研究の SGmonitor と比較した．SGmonitor では，OCALL を呼び出してからハイパーコールを実行することにより，監視対象 VM の OS データを取得する．**図 8** に示すように，SCwatcher のプロセス一覧の取得時間は SGmonitor の 3.4 倍となった．このことから，SCONE を用いて pread システムコールを実行するオーバヘッドまたは OS データを取得する際に VM メモリデバイスを経由することによるオーバヘッドが大きいことが分かった．

そこで，VM メモリデバイスのオーバヘッドを調べるために，ハイパーコールの代わりに VM メモリデバイスにアクセスして OS データを取得するようにした SGmonitor を用いてプロセス一覧の取得を行った．ハイパーコールを用いる従来の SGmonitor と実行時間を比較した結果を**図 9** に示す．VM メモリデバイスを経由する場合の実行時間は，ハイパーコールを直接呼び出した場合の 1.2 倍となった．これにより，VM メモリデバイスを用いるオーバヘッドは小さいことが分かった．つまり，SCONE の非同期システムコール機構を用いてシステムコールを実行することによるオーバヘッドが大きいと考えられる．

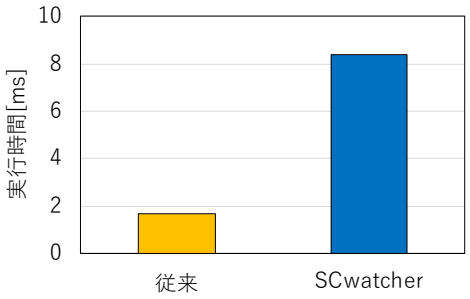

図 7 netstat の実行時間

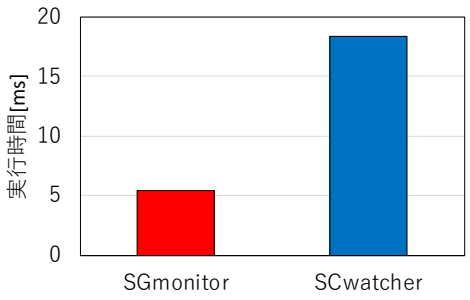

図 8 プロセス一覧の取得時間

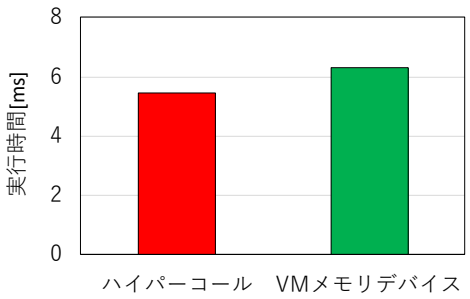

図 9 VM メモリデバイスのオーバヘッド

6. 関連研究

GLvisor [21] はライブラリ OS を用いてエンクレイヴ内で既存の IDS を動かすことを可能にするシステムである．オープンソースの SGX 用ライブラリ OS である Graphene-SGX が IDS に OS の標準インタフェースを提供する．GLvisor は Graphene-SGX の proc ファイルシステム内で OCALL を呼び出すことにより，監視対象 VM から

OS データを取得する．しかし, 実際には Graphene-SGX で既存のアプリケーションを動かすのは難しいため，GLvisor では proc ファイルシステムを用いた簡単なプログラムしか動かすことができていない．

そこで，Graphene-SGX の代わりに SGX-LKL [22] を用いることが考えられる．SGX-LKL もオープンソースであり，Linux をベースとしたライブラリ OS がエンクレイヴ内のアプリケーションに標準 C ライブラリを提供する．SGX-LKL は実際に既存のアプリケーションを動かすことが可能である．このライブラリ OS には proc ファイスシステム以外のファイルシステムやネットワークスタックも含まれており，エンクレイヴとホスト OS 間のインタフェースを最小限にすることができる．しかし，その分だけエンクレイヴ内で動かすコード量が増加し，セキュリティの低下や性能低下につながる恐れがある．

S-NFV [23] はネットワーク機能仮想化（NFV）において，仮想ネットワーク機能の状態とその状態を扱うコードをエンクレイヴ内に移動させる．例として，ネットワーク IDS の Snort ではネットワークフローごとの状態を安全に扱えるようにしている．S-NFV ではエンクレイヴ内で動かすコードを最小限にすることができるが，攻撃を受けないように NFV アプリケーションをうまく 2 つに分割する必要がある．

Transcall [24] は，既存の IDS を VM の外にオフロード可能にするシステムである．IDS に対して VM を監視するための VM シャドウと呼ばれる実行環境を提供する．VM シャドウはライブラリやシステムコール，proc ファイルシステムなどについて監視対象 VM と同じ OS インタフェースを提供する．いくつかのシステムコールと proc ファイルシステムについては監視対象 VM のシステム情報を提供する．また，IDS が監視対象 VM のファイルシステムにアクセスすることも可能にする．

一方，VMST [25] は監視用 VM に既存の IDS をオフロードする．監視用 VM では監視対象 VM と同一のシステムを動かすことにより，IDS に監視対象 VM と同じ OS インタフェースを提供する．監視用 VM の OS は必要に応じて監視対象 VM のメモリから透過的に情報を取得し，それを IDS に提供する．しかし，これらの IDS は外部の攻撃者や信頼できないクラウド内の管理者からの攻撃を受ける可能性がある．

既存の IDS を安全に実行するシステムとして，RemoteTrans [13] が提案されている．RemoteTrans は，IDS をクラウド外部の信頼できるリモートホストにオフロードすることで安全に VM を監視する．IDS はクラウド内のハイパーバイザと暗号通信を行い，監視対象 VM のシステム情報を取得する．しかし，クラウド外部に監視用のリモートホストが必要である．V-Met [14] は，ネストした仮想化を用いることで仮想化システム全体を VM 内で動作させ，この VM の外で既存の IDS を安全に実行するシステムである．しかし，ネストした仮想化のオーバヘッドにより仮想化システムの性能が大幅に低下する．

7. まとめ

本稿では，Intel SGX 向け実行環境である SCONE を利用して OS の標準インタフェースを提供することで，既存の IDS をエンクレイヴ内にオフロードして安全に実行可能にするシステム SCwatcher を提案した．SCwatcher では，エンクレイヴ内 proc ファイルシステムがホスト OS 内の VM メモリデバイスを経由して監視対象 VM のメモリから OS データを取得し，IDS に透過的に提供する．IDS のコンパイル時に標準ファイル関数の呼び出しを置換することにより，必要に応じてエンクレイヴ内の proc ファイルシステムにアクセスさせる．我々は SCwatcher を Xen-SGX 4.7 に実装した．実験の結果，既存の netstat コマンドをエンクレイヴ内で動かして監視対象 VM のネットワーク接続状況が取得できることを確認した．また，SCONE を用いて VM メモリデバイスにアクセスするためのシステムコールを実行することによるオーバヘッドが大きいことが分かった．

今後の課題は，VM メモリデバイスを呼び出すオーバヘッドを削減することである．監視対象 VM のメモリから一度に取得するデータ量を増やすことでオーバヘッドを削減できる可能性がある．また，様々な IDS を実行できるようにすることも今後の課題である．特に exec システムコールによる外部プログラムの実行をどのように実現するかを検討する必要がある．

参考文献

[1] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proceedings of 10th Annual Network and Distributed System Security Symposium*, pp. 191–206 (2003).
[2] 中野智晴，光来健一：Intel SGX を用いた VM のメモリとディスクの安全な監視，コンピュータセキュリティシンポジウム 2019 (2019).
[3] Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P. and Fetzer, C.: Secure Linux Containers with Intel SGX, *Proceedings of the 12th USENIX Symposium on Operating System Design and Implementation*, pp. 689–703 (2016).
[4] Intel Corp: Xen SGX Virtualization Support, https://github.com/intel/xen-sgx/.
[5] Ozaki, Y., Kanamoto, S., Yamamoto, H. and Kourai, K.: Detecting System Failures with GPUs and LLVM, *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, pp. 47–53 (2019).
[6] TechSpot News: Google FIred Employees for Breaching User Privacy, https://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html (2010).

[7] PwC: US Cybercrime: Rising Risk, Reduced Readiness (2014).
[8] CyberArk Software: Global IT Security Service (2009).
[9] Intel Corp: Intel Software Gurad Extensions, https://software.intel.com/en-us/sgx/sdk/.
[10] Li, C., Rughunathan, A. and Jha, N. K.: A Trusted Virtual Machine in an Untrusted Management Environment, *IEEE Transactions on Services Computing*, Vol. 5, No. 4, pp. 472–483 (2012).
[11] Li, C., Rughunathan, A. and Jha, N. K.: Secure Virtual Machine Execution under an Untrusted Management OS, *Proceedings of the 3rd IEEE International Conference on Cloud Computing*, pp. 172–179 (2010).
[12] Santos, N., Gummadi, K. P. and Rodrigues, R.: Towards Trusted Cloud Computing, *Proceedings of Conference on Hot Topics in Cloud Computing* (2009).
[13] Kourai, K. and Juda, K.: Secure Offloading of Legacy IDSes Using Remote VM Introspection in Semi-trusted Clouds, *Proceedings of the 9th IEEE International Conference on Cloud Computing*, pp. 43–50 (2016).
[14] Miyama, S. and Kourai, K.: Secure IDS Offloading with Nested Virtualization and Deep VM Introspection, *Proceedings of the 22th European Symposium on Research in Computer Security*, pp. 305–323 (2017).
[15] Butt, S., Lagar-Cavilla, H. A., Srivastava, A. and Ganapathy, V.: Self-service Cloud Computing, *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pp. 253–264 (2012).
[16] Wang, J., Stavrou, A. and Ghosh, A.: HyperCheck: A Hardware-assisted Integrity Monitor, *Proceedings of the 13th International Conference on Recent Advances in Intrusion Detection*, pp. 158–177 (2010).
[17] Rutkowska, J., Wojtczuk, R. and Tereshkin, A.: HyperGuard, *Xen 0wning Trilogy, Black Hat USA* (2008).
[18] McCune, J. M., Parno, B. J., Perrig, A., Reiter, M. K. and Isozaki, H.: Flicker: An Execution Infrasturcture for TCB Minimization, *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, pp. 315–328 (2008).
[19] Azab, A. M., Ning, P., Wang, Z., Jiang, X., Zhang, X. and Skalsky, N.: HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity, *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pp. 38–49 (2010).
[20] wolfSSL Inc: wolfSSL Embedded SSL/TLS Library, https://www.wolfssl.com/.
[21] 篠原悠介：Intel SGX とライブラリ OS を用いた IDS オフロード，九州工業大学卒業論文 (2019).
[22] Priebe, C., Muthukumaran, D., Lind, J., Zhu, H., Cui, S., Sartakov, V. A. and Pietzuch, P.: SGX-LKL: Securing the Host OS Interface for Trusted Execution, *arXiv:1908.11143* (2019).
[23] 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).
[24] 飯田貴大，光来健一：VM Shadow: 既存 IDS をオフロードするための実行環境，第 119 回 OS 研究会 (2011).
[25] Fu, Y. and Lin, Z.: Space Traveling across VM: Automatically Bridging the Semantic Gap in Virtual Machine Introspection via Online Kernel Data Redirection, *Proceedings of 2012 IEEE Symposium on Security and Privacy*, pp. 586–600 (2012).