Vol.2022-OS-154 No.7
2022/3/15

SGX向け実行環境OcclumとSCONEを用いたVMの安全な監視手法

河村 拓実[1]　光来 健一[1]

概要：IaaS 型クラウドが提供する仮想マシン（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）を用いて監視する必要がある．ホストベース IDS を安全に実行するために，IDS を VM の外で実行する IDS オフロードと呼ばれる手法が用いられているが，オフロードした IDS も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 を用いた安全な IDS オフロードが提案されている．しかし，IDS の開発には OS カーネルレベルのプログラミングが必要となり，SGX 専用ライブラリを用いなければならない．本稿では，SGX 向け実行環境を用いて OS 上で動作する従来の IDS をエンクレイヴ内にオフロードすることを可能にする SCwathcer を提案する．SCwatcher では，SGX 向け実行環境がエンクレイヴ内の IDS に標準的な OS インタフェースを提供する．さらに，SCwatcher は VM のシステム情報を取得可能な VM 監視用 proc ファイルシステムを IDS に提供する．SGX 仮想化をサポートした Xen-SGX および，Occlum と SCONE の 2 種類の SGX 向け実行環境を用いて SCwatcher を実装し，従来の IDS の動作確認と性能比較を行った．

1. はじめに

近年，ユーザに仮想マシン（VM）を提供する IaaS 型クラウドの普及が進んでいる．クラウド内の VM はインターネット経由での攻撃を受けやすいため，侵入検知システム（IDS）を用いて VM を監視することが重要になっている．通常，ホストベース IDS は監視対象 VM の中で実行されるため，攻撃者に VM へ侵入されてしまうと IDS を無効化される恐れがある．そこで，IDS を監視対象 VM の外部で実行する IDS オフロード [1] と呼ばれる手法が用いられている．この手法を用いることにより，攻撃者が VM に侵入したとしても IDS は VM 内には存在しないため無効化される恐れはない．しかし，オフロードした IDS もまた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 が攻撃される恐れががあるからである．また，クラウド内に内部犯がいる可能性もある．

この問題を解決するために，Intel 製 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 SGmonitor [2] が提案されている．SGmonitor では，SGX によって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．しかし，SGmonitor 上で動作する IDS を開発するのは容易ではない．オフロードした IDS は VM のメモリ上の OS データを解析して監視を行うため，OS カーネルレベルのプログラミングが必要となるためである．これはアプリケーションレベルのプログラミングと比べて難しく，OS のバージョンの影響も大きく受ける．また，エンクレイヴ内では SGX 専用ライブラリを用いなければならないため，従来とは異なるプログラミングを行う必要がある．

本稿では，SGX 向け実行環境を用いてエンクレイヴ内に従来の IDS をオフロード可能にするシステム SCwatcher を提案する．SGX 向け実行環境を用いることで，エンクレイヴ内の IDS は標準 C ライブラリやシステムコールなどの OS の標準インタフェースを利用することができる．さらに，SCwatcher は監視対象 VM のシステム情報を取得可能な proc ファイルシステムを IDS に提供する．この VM 監視用 proc ファイルシステムは信頼するハイパーバイザ経由で VM のメモリデータを取得し，疑似ファイルを作成する．IDS はこの疑似ファイルにアクセスすることで VM の情報を取得することができる．これにより，IDS は VM 内と同じ OS インタフェースを利用することができるため，既存の IDS をオフロードして実行することが可能になる．IDS を新規開発する際も，従来の IDS と同じように開発することができる．

我々は SGX 仮想化をサポートした Xen-SGX 4.7 [3] を用いて SCwatcher を実装した．SGX 向け実行環境には様々

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

なものがあるが，性能やセキュリティなどの面でトレードオフがある．そこで，SCwatcher は Occlum [4] と SCONE [5] の 2 種類の実行環境をサポートしている．Occlum を用いる場合は，実行環境に含まれる proc ファイルシステムを拡張することで VM 監視用 proc ファイルシステムを実装した．VM のメモリデータを取得する際には，SGX の OCALL を用いてエンクレイヴ外で動作するランタイムを呼び出し，そこからハイパーバイザを呼び出す．一方，SCONE を用いる場合はオープンソースではないため，VM 監視用 proc ファイルシステムを独自に実装した．そして，システムコールのインタフェースを用いてエンクレイヴ外の OS を呼び出し，そこからハイパーバイザを呼び出す．IDS が VM 監視用 proc ファイルシステムを呼び出せるようにするために，IDS のコンパイル時にプログラム変換を行う．

SCwatcher を用いてシェルスクリプトと外部コマンドを用いる既存の IDS である chkrootkit [6] を実行し，VM のネットワークとプロセスの監視が行えることを確認した．また，chkrootkit の実行時間を測定し，SCwatcher のオーバヘッドを調査した．その結果，SCwatcher では外部コマンドを呼び出すオーバヘッドが非常に大きいことが分かった．そこで，外部コマンドを用いずに chkrootkit と同等の検査機能を実現する IDS を Python および C 言語を用いて実装して性能を測定した．その結果，C 言語版 chkrootkit は従来の安全ではない IDS オフロードに近い性能を達成できることが分かった．

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 を用いた安全な IDS オフロードについて述べる．3 章では SGX を用いて従来の 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 の外部で安全にホストベース IDS を実行するための手法である．この手法を用いることにより，攻撃者が VM 内に侵入したとしても IDS は VM の内部には存在しないため，無効化される恐れはない．監視対象 VM 内で実行される IDS とは異なり，オフロードした IDS は VM のメモリデータを取得し，その中の OS データを解析することによって監視を行う．例えば，監視対象 VM 内のネットワークの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．また，実行されているプロセスの一覧を調べることで，マルウェアの実行を検知することができる．

しかし，IDS は VM の外部にオフロードしたとしてもまだ攻撃を受ける恐れがある．クラウド外部の攻撃者から攻撃を受ける可能性があるためである．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 [7]．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 [8] 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 [9] もある．オフロードした IDS が攻撃を受けると監視が正常に行えなくなったり，監視対象 VM から取得した機密情報が漏洩したりする危険がある．

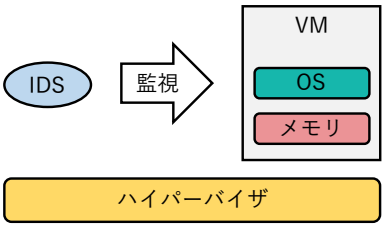

図 1 IDS オフロード

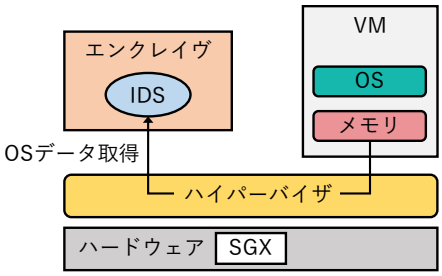

図 2 SGmonitor のシステム構成

この問題を解決するために，Intel SGX を用いて安全な IDS オフロードを実現する SGmonitor [2] が提案されている．SGX は Intel 製 CPU が提供するセキュリティ機構であり，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，データの漏洩も防ぐことができる．SGmonitor は**図 2** のように，IDS をエンクレイヴ内にオフロードすることで IDS が攻撃を受けないように保護する．SGmonitor は VM の下で動くハイパーバイザを信頼し，ハイパーバイザ経由で VM から監視に必要なメモリデータを安全に取得する．

しかし，SGmonitor 上で動作する IDS の開発には OS カーネルレベルのプログラミングが必要となる．IDS は OS の内部構造に関する情報を用いて監視対象 VM のメモリデータを解析することで，監視に必要な OS データを取得するためである．このような IDS の開発は OS カーネル内で動作する IDS 以外では行われておらず，OS 上で動作する一般的な IDS の開発に比べて難しい．その上，監視対

象 VM 内の OS に強く依存するため，OS のバージョンごとに IDS を開発する必要がある．また，エンクレイヴ内で IDS を実行するには Intel SGX SDK [10] などの専用ライブラリを用いなければならない．そのため，従来とは異なるプログラミングを行う必要がある．

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 [11], [12], [13], [14], [15], [16]．クラウド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 のハードウェアおよびエンクレイヴ内で動作するソフトウェアには脆弱性がないことを仮定する．また，クラウド内のハイパーバイザも信頼する．ハイパーバイザが正常に動作していることは様々な既存手法を用いて確認することができる．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改竄されていないハイパーバイザが起動したことを確認することができる．システム管理モード（SMM）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 [17], [18], [19], [20]．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 向け実行環境を用いた IDS オフロード

SCwatcher は IDS に OS の標準インタフェースを提供することで，従来の IDS をエンクレイヴ内にオフロードして実行可能にする．そのために，エンクレイヴ内で既存のアプリケーションを実行可能にする SGX 向け実行環境を用いる．この実行環境を用いることにより，IDS はエンクレイヴ内でも標準 C ライブラリやシステムコールなどの従来のインタフェースを利用することができる．**図 3** に示すように，SGX 向け実行環境はエンクレイヴ内で動作するライブラリとエンクレイヴ外で動作するランタイムからなる．IDS は必要に応じてライブラリおよびランタイム経由でエンクレイヴ外のホスト OS の機能を呼び出すことができる．

SCwatcher ではそれに加えて，エンクレイヴ内の IDS に VM 監視用の proc ファイルシステムを提供する．proc ファイルシステムはシステム情報を取得するためのインタフェースであり，IDS の多くが監視に必要な情報を取得するために標準的に利用している OS の機能である．IDS は VM 監視用 proc ファイルシステムの疑似ファイルにアクセスすることで，VM 内のシステム情報を透過的に取得することができる．疑似ファイルはディスク上に格納される通常ファイルと異なり，アクセスされた時に動的にデータが生成される特殊なファイルである．例えば，/proc/プロセス ID/stat という疑似ファイルはそのプロセスの状態やメモリの使用状況などの情報を提供する．

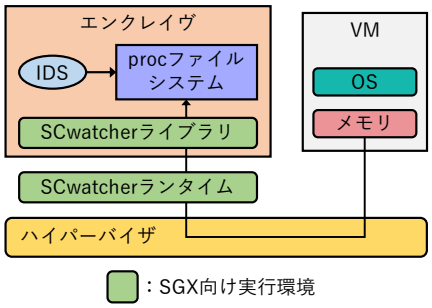

図 3 SCwatcher のシステム構成

VM 監視用 proc ファイルシステムは IDS によって疑似ファイルがオープンされる時に，VM のメモリ上にある OS データを取得する．まず，ランタイム経由でハイパーバイザを呼び出し，対象の OS データが含まれるメモリのデータを取得する．ハイパーバイザ内でメモリデータを暗号化し，エンクレイヴ内で復号することにより，これら以外のランタイムなどへの情報漏洩を防ぐ．そして，プロセス構造体やメモリ構造体などの OS のデータ構造を解析し，疑似ファイルのデータを生成する．IDS によって疑似ファイルが読み込まれる際には，オープン時に生成したファイルデータの中の要求された部分を返す．

SGX 向け実行環境として様々なものが提案されているが，性能やセキュリティなどの面でトレードオフがある．そこで，SCwatcher は Occlum [4] と SCONE [5] の 2 種類の実行環境をサポートしている．これらはともに標準 C ライブラリを提供し，Occlum ではライブラリ OS も提供している．そのため，Occlum の方が多機能であるが，SCONE の方が Trusted Computing Base (TCB) が小さいため，脆弱性を含んでいる可能性が低い．Occlum では MMDSFI と呼ばれる機能を用いて隔離することにより，1 つのエンクレイヴ内で複数のプロセスを安全に実行可能であるが，各プロセスは共有するライブラリ OS 経由で影響を受ける可能性がある．ただし，ライブラリ OS の大部分は Rust で記述されているため，脆弱性は少ないと考えられる．一方，SCONE ではエンクレイヴ内で 1 つのプロセスしか実行することができないため，子プロセスを作成したり新しいプログラムを実行したりするたびにエンクレイヴを作成する必要がある．そのためのオーバヘッドは大きいが，プロセス間の隔離は Occlum より強い．

SCwatcher において Occlum を用いる場合，ライブラリ OS の中に proc ファイルシステムが含まれているため，このファイルシステムを拡張することで VM 監視用 proc ファイルシステムとする．監視対象 VM の OS データを取

得する際には，まず，OCALL を用いてエンクレイヴ外部の Occlum 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安全に呼び出すための SGX のインタフェースである．そして，Occlum ランタイムを拡張することでハイパーバイザを呼び出し，VM のメモリデータを取得する．

一方，SCONE を用いる場合は，SCONE ライブラリに proc ファイルシステムは含まれていないため，独自の VM 監視用 proc ファイルシステムを提供する．Occlum とは異なり，SCONE のソースコードは公開されていないため，SCONE ライブラリや SCONE ランタイムを拡張して監視対象 VM の OS データを取得できるようにすることはできない．そのため，SCONE が提供するシステムコール呼び出しのインタフェースを用いて，ホスト OS 内に用意したデバイスにアクセスする．そして，そのデバイスがハイパーバイザを呼び出して VM のメモリデータを取得する．この VM 監視用 proc ファイルシステムは IDS にリンクされるが，リンクするだけでは IDS に使わせることはできない．IDS は通常通り，SCONE ライブラリ経由でホスト OS の proc ファイルシステムにアクセスしてしまうためである．そこで，IDS のコンパイル時にプログラム変換を行うことで，エンクレイヴ内の VM 監視用 proc ファイルシステムにアクセスさせる．

4. 実装

我々は SGX 仮想化をサポートしたハイパーバイザの Xen-SGX 4.7 [3] を用いて SCwatcher を実装した．Xen-SGX ではドメイン 0 の中にエンクレイヴを作成できないため，IDS 専用の VM（IDS VM）の中にエンクレイヴを作成して IDS をオフロード実行した．SGX 向け実行環境として Occlum と SCONE の 2 種類を用いた．

4.1 Occlum を用いた実装

Occlum を用いる場合，**図 4** のように Occlum ライブラリ内の proc ファイルシステムを拡張し，VM 監視用 proc ファイルシステムを実装した．この proc ファイルシステムは LLView [2] を用いることにより，VM 内の OS のソースコードを用いて疑似ファイルのデータを生成する．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，カーネル変数，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フレームワークである．LLView はプログラムをコンパイルして生成される LLVM の中間表現に対してプログラム変換を行う．これにより，proc ファイルシステムが VM 内の OS データにアクセスしようとした時に，自動的に VM のメモリ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 VM 内のカーネルのメモリデータを取得する g_map 関数の呼び出しが挿入される．g_map 関数はまず，取得する OS データの仮想アドレスのページオフセット（下位 12 ビット）を 0 にし，ページの先頭アドレス（ページアドレス）を取得する．次に，ページアドレスと監視対象となる VM のドメイン ID を引数として OCALL を実行し，エンクレイヴ外の Occlum ランタイムを呼び出す．そして，Occlum ランタイムがハイパーコールを実行し，指定したページのメモリデータを取得する．

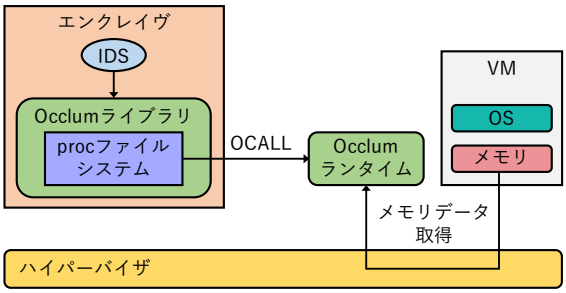

図 4 Occlum を用いた SCwatcher

一方，proc ファイルシステムが VM 内のプロセスのメモリデータを取得する際には，明示的に g_proc_map 関数を実行する．例えば，プロセスの実行時のコマンドラインはプロセスのメモリに格納されている．g_proc_map 関数は g_map 関数と同様に，取得するプロセス内のデータの仮想アドレスからページアドレスを取得する．このページアドレスに加えて，対象プロセスのページテーブルの仮想アドレスと VM のドメイン ID を OCALL の引数として Occlum ランタイムを呼び出し，ハイパーコールを実行する．

4.2 SCONE を用いた実装

SCONE を用いる場合，独自の VM 監視用 proc ファイルシステムを実装した．この proc ファイルシステムは Occlum を用いる場合と同様に LLView を用いて実装した．

4.2.1 VM メモリデバイスへのアクセス

図 5 に示すように，VM 監視用 proc ファイルシステムに対して VM のメモリへのインタフェースを提供する VM メモリデバイスをホスト OS 内に用意した．VM メモリデバイスは疑似デバイスとして作成し，デバイスのマイナー番号で監視対象となる VM のドメイン ID を指定する．LLView が自動挿入する g_map 関数や明示的に呼び出す g_proc_map 関数内で，このデバイスに pread システムコールを用いてアクセスすることで VM のメモリデータを取得する．pread は引数で指定したオフセットからファイルデータを読み込むシステムコールである．

g_map 関数は OS データのページアドレスをファイルオフセットとして引数に指定して pread を実行する．しかし，カーネルのページアドレス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，**図 6** のようにして圧縮を行う．カーネルのページアドレスの下位 12 ビットは

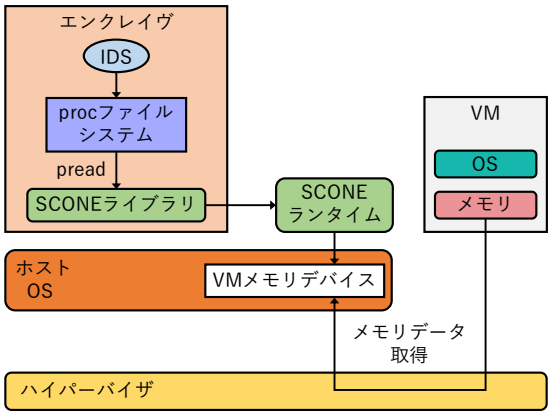

図 5 SCONE を用いた SCwatcher

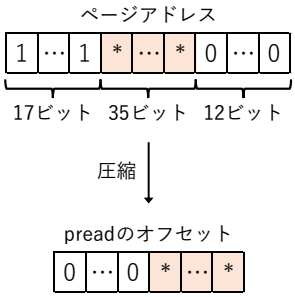

図 6 g_map 関数におけるアドレスの圧縮

常に 0 であり，上位 17 ビットは常に 1 となるため，これらを VM メモリデバイスに渡す必要はない．そこで，残りの 35 ビットを取り出してファイルオフセットの下位 35 ビットとし，残りのビットを 0 にすることで正常な範囲のファイルオフセットとする．

一方，g_proc_map 関数では，プロセスデータのページアドレスに加えて，対象プロセスのページテーブルのアドレスも VM メモリデバイスに渡す必要がある．そこで，これら 2 つのアドレスを**図 7** のように 1 つのファイルオフセットに圧縮する．プロセスデータのページアドレスについては，g_map 関数の場合と同様にファイルオフセットの下位 35 ビットに圧縮する．さらに，ページテーブルの仮想アドレスを物理アドレスに変換し，その下位 40 ビットのみを用いる．それにより VM に割り当て可能なメモリは 1TB に制限されるが，一般的には十分なメモリサイズであると考えられる．ページテーブルの物理アドレスの下位 12 ビットも 0 であるため，残りの 28 ビットをファイルオフセットの次の 28 ビットとする．最上位ビットはファイルオフセットが不正な値とならないように 0 にする．

pread システムコールを実行すると VM メモリデバイスが呼び出され，読み出し処理を行う関数が実行される．この関数では，VM メモリデバイスのマイナー番号を取得することにより監視対象 VM のドメイン ID を特定する．その後，引数で受け取ったファイルオフセットの値から，対象データのページアドレスとページテーブルのアドレスを復元する．ページテーブルのアドレスが 0，つまり g_map

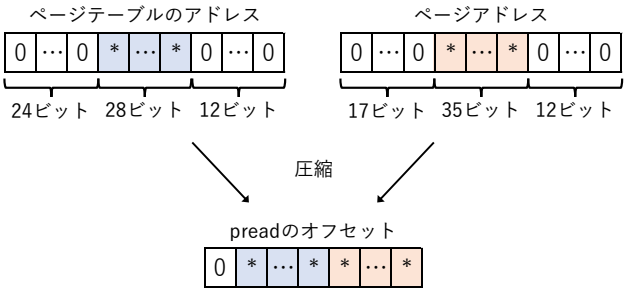

図 7 g_proc_map 関数におけるアドレスの圧縮

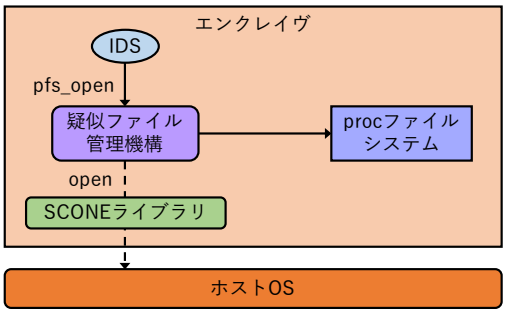

図 8 疑似ファイル管理機構

関数から呼び出された場合は，VM 内のカーネルのメモリデータを取得するハイパーコールを実行する．一方，ページテーブルのアドレスが 0 以外，つまり g_proc_map 関数によって呼び出された場合は，VM 内のプロセスのメモリデータを取得するハイパーコールを実行する．

4.2.2 疑似ファイル管理機構

IDS が標準ファイル関数を呼び出す際には，SCONE ライブラリではなく**図 8** 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，LLView をベースにしたツールである REPLLVM を開発し，IDS のコンパイル時にプログラム変換を行うようにした．REPLLVM は中間表現において call 命令で呼び出される標準ファイル関数を pfs_ というプレフィックスの付いた関数に置換する．それにより，疑似ファイル管理機構が必要に応じてエンクレイヴ内の VM 監視用 proc ファイルシステムを呼び出す．疑似ファイル管理機構は疑似ファイルを PFILE 構造体を用いて管理する．この構造体は，疑似ファイルのデータを格納するバッファや疑似ファイルのサイズ，現在のファイルオフセットを管理する．/proc などの疑似ディレクトリは，dirent 構造体のリストを保持する PDIR 構造体を用いて管理する．

例えば，IDS が open 関数を実行しようとすると，pfs_open 関数が呼び出される．pfs_open 関数はまず，引数のファイルパスを調べて，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．通常ファイルの場合は，標準ファイル関数の open 関数を実行し，その返り値を呼び出し元に返す．疑似ファイルの場合は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，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ことで，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．

次に，VM 監視用 proc ファイルシステムを呼び出して疑似ファイルのデータを生成し，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，pfs_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．pfs_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．

疑似ディレクトリをオープンする pfs_opendir 関数は，引数のパスが/proc で始まっていれば，task_struct 構造体をたどって VM 内のプロセス ID の一覧を取得する．そして，/proc ディレクトリの中のエントリを作成し，固定アドレスに疑似ファイル番号とは別の疑似ディレクトリ番号を足したアドレスを DIR 構造体のポインタとして返す．現在のところ，疑似ディレクトリの操作は/proc ディレクトリにのみ対応している．

pfs_read 関数などの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）を引いた値を計算する．これが疑似ファイル番号の範囲外であれば，ディスクリプタが指すのは通常ファイルであるため対応する標準ファイル関数を呼び出す．疑似ファイルであった場合は，疑似ファイル番号をインデックスとして PFILE 構造体の配列を参照し，疑似的なファイル処理を行う．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．

4.3 VM のメモリデータの取得

VM 内のカーネルのメモリデータを取得するために，ハイパーバイザにハイパーコールを追加した．このハイパーコールを実行すると，呼び出されたハイパーバイザが VM の仮想 CPU から CR3 レジスタの値を取得する．そのアドレスに基づいて 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メモリ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 のメモリデータをページ単位で返す．

さらに，VM 内で動作しているプロセスのメモリデータを取得可能にするために，別のハイパーコールを追加した．このハイパーコールはメモリデータのアドレスに加えて，メモリに関する情報が格納されている mm_struct 構造体から取得した対象プロセスのページテーブルのアドレスも指定して実行する．呼び出されたハイパーバイザは，指定されたページテーブルを用いてアドレス変換を行い，プロセスのメモリデータをページ単位で返す．

VM のメモリデータを取得する過程で盗聴されるのを防ぐために，取得したデータはハイパーバイザ内で暗号化する．暗号化には組み込み向け SSL/TLS ライブラリである wolfSSL [21] を用いた．暗号化したデータはエンクレイヴ内で復号し，ハッシュ表を用いてキャッシュする．VM 監視用 proc ファイルシステムからハイパーバイザに渡す仮想アドレスも同様にして暗号化する必要がある．また，改竄を検知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．これらの機能は SGmonitor には実装されているが，SCwatcher には現在のところ未実装である．

4.4 VM のファイルへのアクセス

IDS が VM のファイルにアクセスできるようにするために，SCwatcher では監視対象 VM の仮想ディスクを IDS VM の/tmp/vm にマウントする．マウントするには IDS VM から監視対象 VM のイメージファイルにアクセスする必要があるが，このイメージファイルはドメイン 0 に存在するため，NFS を利用して IDS VM とドメイン 0 間でファイルを共有する．

Occlum を用いる場合，IDS VM 上に作られる Occlum 用のコンテナ内で IDS を起動するため，コンテナの起動時に IDS VM の/tmp/vm をコンテナの/root/vm にマウントする．そして，Occlum ライブラリ内で open システムコールと stat システムコールをフックし，引数のパス名の先頭に/root/vm を追加することで仮想ディスクへリダイレクトする．

SCONE を用いる場合，REPLLVM を用いて標準ファイル関数を”vdisk_”というプレフィックスの付いた仮想ディスクアクセス関数に置換する．仮想ディスクアクセス関数は，引数のパス名の先頭に/tmp/vm を追加し，このパスを引数にして標準ファイル関数を実行する．

5. 実験

SCwatcher を用いて既存の IDS である chkrootkit をオフロード実行した．chkrootkit はシェルスクリプトで記述されているため，エンクレイヴ内で bash を実行し，その上で chkrootkit を動かした．この実験では，Occlum または SCONE を用いて実装した SCwatcher と，SGX を用いない従来の安全ではない IDS オフロードの性能を比較した．Occlum を用いた SCwatcher では，事前に IDS 用に作成しておいたエンクレイヴ内で chkrootkit を実行した．一般に，IDS は定期的に実行されるため，このエンクレイヴ内で一定時間ごとに IDS を実行するスクリプトを動作させ

```
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
```

図 9 netstat の実行結果

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.3 119964  6188 ?        Ss   Feb02   0:00 /sbin/init splash
root          2  0.0  0.0      0     0 ?        S    Feb02   0:00 [kthreadd]
```

図 10 ps の実行結果

```
Checking `slapper'... Warning
Checking `inetd'... INFECTED
Checking `sshd'... INFECTED
Checking `tcpd'... INFECTED
```

図 11 chkrootkit の実行結果

れば，エンクレイヴを作成するのは最初の 1 回だけで済むためである．

実験に使用したマシンの CPU は Intel Core i7-8700，メモリは 16GB，HDD は 2TB であった．IDS VM と監視対象 VM にはそれぞれ仮想 CPU を 2 個，メモリを 2GB，仮想ディスクを 80GB または 50GB 割り当てた．

5.1 既存 IDS のオフロード実行の動作

まず，chkrootkit で用いられる外部コマンドの内，proc ファイルシステムにアクセスを行う netstat コマンドと ps コマンドを SCwatcher 上で実行した．netstat はネットワークの接続状況を取得するコマンドであり，ps は実行中のプロセスに関する情報を取得するコマンドである．これらのコマンドの実行結果の一部を **図 9** と **図 10** に示す．この出力結果は VM 上でコマンドを実行した場合とほぼ一致することを確認した．

次に，chkrootkit において proc ファイルシステムにアクセスを行う 1 つのネットワーク検査機能と 3 つのプロセス検査機能を実行した．ネットワークの検査では，slapper と呼ばれるワームに感染していないかのチェックを行う．プロセスの検査では，inetd，sshd，tcpd の 3 つのデーモンプロセスが不正なものでないかをチェックする．VM を疑似的にマルウェアに感染させた状態で chkrootkit を実行したところ，**図 11** に示す実行結果が得られ，正常にマルウェアを検知できることを確認した．

5.2 既存の外部コマンドの実行時間

SCwatcher 上で netstat コマンドと ps コマンドの実行にかかる時間を測定した．**図 12** に実験結果を示す．Occlum を用いた SCwatcher では，SCONE を用いた場合の 5.8~11

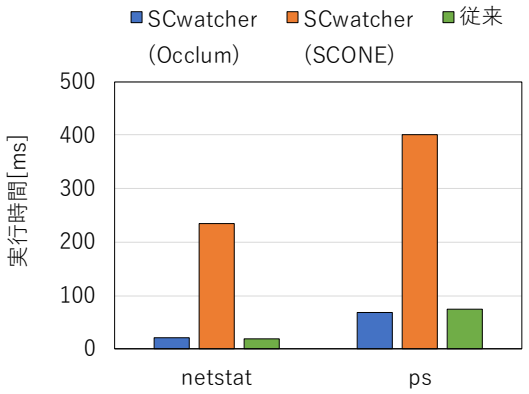

図 12 既存の外部コマンドの実行時間

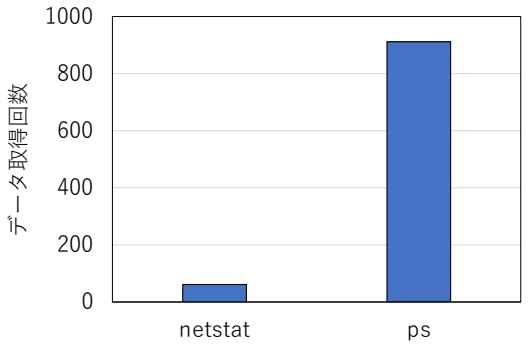

図 13 VM のメモリデータの取得回数

倍高速に実行できることが分かった．これは，Occlum を用いた場合にはコマンド実行時に新たなエンクレイヴを作成しないで済んだためである．ただし，コマンドごとにエンクレイヴを作成する SCONE の方がより安全であると考えられる．このことから，Occlum を用いる場合と SCONE を用いる場合とで性能とセキュリティのトレードオフが取れると考えられる．

一方，Occlum を用いた SCwatcher における実行時間は従来の IDS オフロードの 0.9~1.1 倍となった．**図 13** に示すように，ps コマンドでは SCwatcher のオーバヘッドになると考えられる，VM からのメモリデータ取得の回数が非常に多かった．しかし，従来の IDS オフロードよりもむしろ性能は高くなった．このことから，外部コマンドの実行に関しては Occlum を用いた SCwatcher のオーバヘッドは小さいことが分かった．

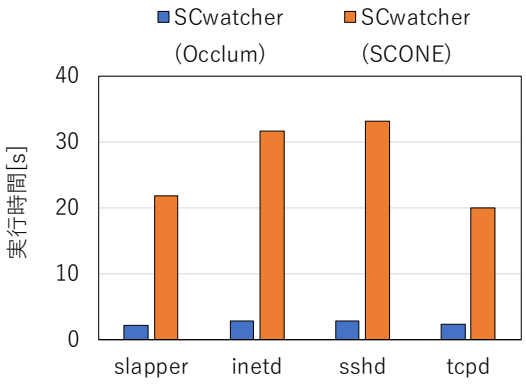

図 14 chkrootkit の実行時間

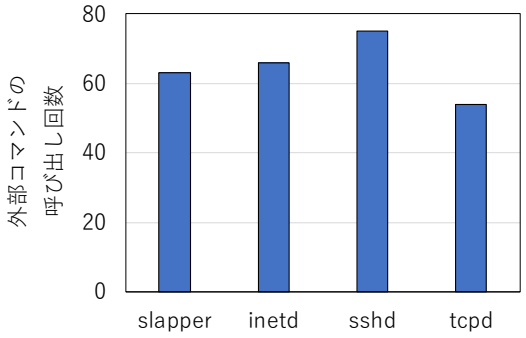

図 15 外部コマンドの呼び出し回数

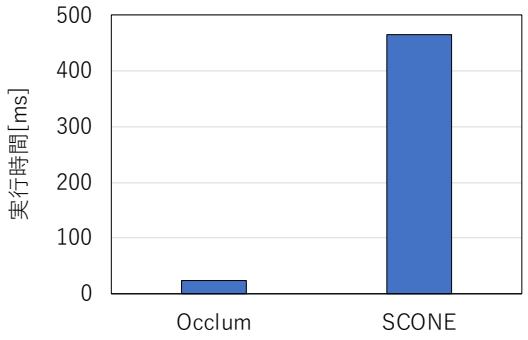

図 16 外部コマンドの呼び出しにかかる時間

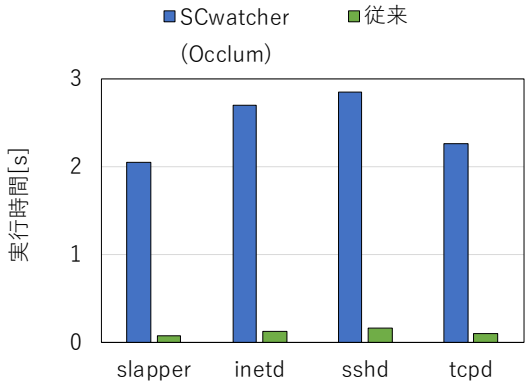

図 17 chkrootkit の実行時間（従来との比較）

5.3 chkrootkit の実行時間

chkrootkit のネットワークとプロセスに関する 4 つの検査機能の実行時間を測定した．**図 14** に示すように，Occlum を用いた SCwatcher は SCONE を用いた場合よりも 8.9~12 倍高速であった．このように差が大きくなった原因を調べるために，外部コマンドの呼び出し回数と呼び出しにかかる平均時間を測定した．それぞれの実験結果を**図 15** と**図 16** に示す．この結果より，chkrootkit では外部コマンドの呼び出し回数が非常に多いことが分かった．また，SCONE を用いた場合，外部コマンドの呼び出しにかかる時間は Occlum の場合の 20 倍であった．SCONE では外部コマンドを呼び出すたびに vfork システムコールと execve システムコールを実行するが，そのたびにエンクレイヴを作成するため，外部コマンドの呼び出しにかかる時間の合計が非常に長くなったと考えられる．Occlum を用いた場合には，chkrootkit の実行時だけでなく，外部コマンドの実行時にもエンクレイヴを作成しなかった．

一方，**図 17** に示すように，Occlum を用いた SCwatcher でも従来の IDS オフロードの 18~29 倍の時間がかかった．Occlum は外部コマンドを呼び出す際に posix_spawn システムコールを実行してエンクレイヴを作成せずに済ませるが，それでもこのシステムコール実行にかかる時間の合計が chkrootkit の実行時間の多くを占めていることが分かった．このことから，SCwatcher は外部コマンドを多用して大量のプロセスを作成する IDS にはあまり向いていないことが分かった．

5.4 Python 版 chkrootkit の実行時間

SCwatcher 上で外部コマンドを用いない IDS を実行する性能を調べるために，chkrootkit の 4 つの検査機能を Python を用いて実装した．この Python 版 chkrootkit は単一プロセスで実行される．Occlum を用いた SCwatcher での実行時間は**図 18** に示すようになり，従来の IDS オフロードでの実行時間の 2.3~2.6 倍になった．この結果から，外部コマンドを多用するシェルスクリプト版 chkrootkit と比べると大幅に性能が改善することが分かった．一方で，Python プログラムは実行時に多くのファイルにアクセスするため，ファイルシステムにアクセスするオーバヘッドが大きい Occlum を用いた SCwatcher の性能はまだ大きく低下することも分かった．

5.5 C 言語版 chkrootkit の実行時間

IDS によるファイルアクセスを減らすために，chkrootkit の 4 つの検査機能を C 言語を用いて実装し，実行時間を測定した．**図 19** に実験結果を示す．Occlum を用いた SCwatcher における実行時間は従来の IDS オフロードの 0.9~1.2 倍となった．既存の外部コマンドを実行する場合と同様に，C 言語で記述された IDS であれば従来の IDS オ

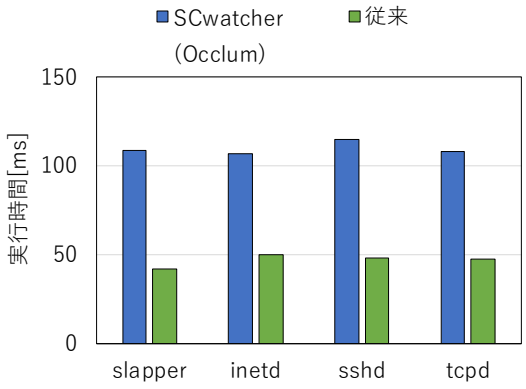

図 18 Python 版 chkrootkit の実行時間

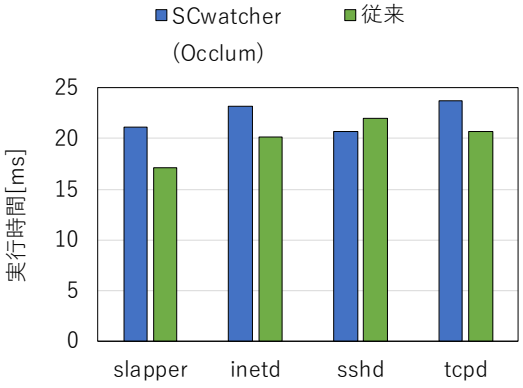

図 19 C 言語版 chkrootkit の実行時間

フロードに近い性能が得られることが分かった．

6. 関連研究

SGX を用いて既存の IDS をオフロードする手法として S-NFV [22] が提案されている．S-NFV で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，既存のネットワーク IDS である Snort において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しつつ NFV アプリケーションをうまく 2 つに分割するのは容易ではない．SCwatcher では既存の IDS 全体をエンクレイヴ内で動作させる．

SEC-IDS [23] はほぼ修正なしに Snort 全体をエンクレイヴ内で実行することができる．そのために，SEC-IDS は SGX 向け実行環境の Graphene-SGX [24] を用いる．それに加えて，DPDK を用いることでエンクレイヴ内でパケットを効率的に取得することができる．Snort の状態がエンクレイヴ・ページキャッシュより小さければ SGX を用いない場合と同等の性能を達成している．しかし，Graphene-SGX は標準 C ライブラリとしてサイズの大きい glibc を用いているため，TCB が大きく攻撃を受けやすい．それに対して，SCwatcher が用いている SCONE や Occlum はサイズの小さい musl を用いている．また，SCwatcher はホストベース IDS を対象としている．

既存の IDS をオフロード可能にするシステムとして Transcall [25] が提案されている．Transcall では，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．

VMST [26] は，監視用 VM に既存の IDS をオフロードすることを可能にする．監視用 VM で監視対象 VM と同一のシステムを動かすことにより，監視対象 VM と同じ OS インタフェースを IDS に提供する．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，IDS に提供する．しかし，Transcall や VMST の IDS は保護されていないため，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．また，これらのシステムをそのままエンクレイヴ内で動作させることはできない．

RemoteTrans [14] 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 を監視するシステムである．IDS はクラウド内のハイパーバイザと暗号通信を行い，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 [15] 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，SGX 向け実行環境の Occlum と SCONE を用いて OS の標準インタフェースを提供することで，従来の IDS をエンクレイヴ内にオフロード可能にする SCwatcher を提案した．SCwatcher 上で動作する IDS はエンクレイヴ内の VM 監視用 proc ファイルシステムにアクセスすることで VM のシステム情報を取得することができる．VM 監視用 proc ファイルシステムはハイパーバイザ経由で監視対象 VM のメモリデータを取得し，IDS に提供する疑似ファイルを作成する．実装した SCwatcher を用いて従来の IDS を実行し，VM の監視が行えることを確認した．また，監視性能を調べた結果，C 言語で記述された IDS は SGX を用いない従来の IDS オフロードに近い性能を達成できることが分かった．

今後の課題は，VM のメモリデータ取得のオーバヘッド

を削減することである．このオーバヘッドは必要となるメモリデータを先読みして取得することで削減できる可能性がある．また，外部コマンドを実行するオーバヘッドも削減する必要がある．加えて，暗号化された VM の仮想ディスクの監視も行えるようにする予定である．そのためには，SGmonitor と同様にエンクレイヴ内で VM のファイルにアクセス可能なファイルシステムを動作させる必要がある．

謝辞 本研究の一部は，JST，CREST，JPMJCR21M4 の支援を受けたものである．

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] Nakano, T. and Kourai, K.: Secure Offloading of Intrusion Detection Systems from VMs with Intel SGX, *Proceedings of the 14th IEEE International Conference on Cloud Computing*, pp. 467–477 (2021).
[3] Huang, K.: Introduction to Intel SGX and SGX Virtualization, Xen Project Developer and Design Summit (2017).
[4] 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 25th International Conference on Architectural Support for Programming Language and Operating Systems*, pp. 955–970 (2020).
[5] 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).
[6] Murilo, N. and Steding-Jessen, K.: chkrootkit – locally checks for signs of a rootkit, `http://www.chkrootkit.org/`.
[7] TechSpot News: Google FIred Employees for Breaching User Privacy, `https://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html` (2010).
[8] PwC: US Cybercrime: Rising Risk, Reduced Readiness (2014).
[9] CyberArk Software: Global IT Security Service (2009).
[10] Intel Corp: Intel Software Gurad Extensions, `https://github.com/intel/linux-sgx/`.
[11] 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).
[12] 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).
[13] Santos, N., Gummadi, K. P. and Rodrigues, R.: Towards Trusted Cloud Computing, *Proceedings of Conference on Hot Topics in Cloud Computing* (2009).
[14] 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).
[15] 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).
[16] 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).
[17] 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).
[18] Rutkowska, J., Wojtczuk, R. and Tereshkin, A.: HyperGuard, *Xen 0wning Trilogy, Black Hat USA* (2008).
[19] 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).
[20] 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).
[21] wolfSSL Inc: wolfSSL Embedded SSL/TLS Library, `https://www.wolfssl.com/`.
[22] 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).
[23] Kuvaiskii, D., Chakrabarti, S. and Vij, M.: Snort Intrusion Detection System with Intel Software Guard Extension (Intel SGX), in arXiv:1802.00508 (2018).
[24] Tsai, C.-C., Porter, D. E. and Vij, M.: Secure Linux Containers with Intel SGX, *Proceedings of the 2017 USENIX Annual Technical Conference*, pp. 645–658 (2017).
[25] 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).
[26] 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).