クラウドにおけるIntel SGXを用いたVMの安全な監視機構

中野 智晴[1]　光来 健一[1]

概要：近年，IaaS 型クラウドの普及が進んでいるが，クラウド内の仮想マシン（VM）はインターネットを経由した攻撃を受けやすい．そのため，侵入検知システム（IDS）を用いて VM を監視することがますます重要となっており，IDS を監視対象 VM の外側で安全に実行する IDS オフロードと呼ばれる手法が提案されている．しかし，IDS オフロードを用いてもクラウド内の信頼できない管理者や外部の攻撃者によってオフロードした IDS が攻撃される恐れがある．これまでに提案されてきた手法では，クラウド内で高度な IDS を安全に実行し，かつ，システム性能への影響を小さくするのは難しかった．そこで本稿では，Intel SGX を用いてクラウド内で IDS を安全に実行し，正しい IDS だけが VM 内の情報を取得できるシステム SGmonitor を提案する．SGmonitor はエンクレイヴと呼ばれる保護領域内で IDS を動作させることにより IDS の改ざんを防ぎ，監視対象 VM から取得した機密情報の漏洩を防ぐことを可能にする．エンクレイヴはアプリケーションの一部であるため，高度な IDS の開発が行いやすく，システム性能に及ぼす影響も小さい．我々は SGX をサポートした Xen 4.7 に SGmonitor を実装し，オフロードした IDS の性能について調べた．

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．ユーザは，必要に応じて様々な OS やソフトウェアを VM 内にインストールできる．一方で，クラウド内の VM はインターネットを経由して攻撃を受けやすいという問題がある．VM が攻撃された場合，VM 内の機密情報が盗まれる恐れがあるため，侵入検知システム（IDS）を用いて VM を監視することがますます重要となっている．IDS は監視対象の VM 内で動作させるのが一般的であるが，VM に侵入した攻撃者に無力化される危険性がある．そこで，監視対象 VM の外側で IDS を安全に実行する IDS オフロードと呼ばれる手法が提案されている [1]．IDS オフロードを用いることによって，VM に侵入したとしても攻撃者が IDS を無効化することはできない．

しかし，クラウドの管理者は必ずしも信頼できるとは限らず，信頼できない管理者からオフロードした IDS への攻撃が行われる危険性がある．また，IDS に対してクラウド外部から攻撃が行われる可能性もある．その結果，オフロードした IDS が取得した VM 内の機密情報を管理者や攻撃者に盗まれる恐れがある．これまでにオフロードした IDS を保護することのできる様々なシステムが提案されてきた [2–10] が，クラウド内で高度な IDS を安全に実行でき，かつ，システム性能に対する影響を小さくするのは難しかった．

本稿では，Intel SGX を用いてクラウド内の IDS を保護することにより，VM を安全に監視できるようにするシステム SGmonitor を提案する．SGX は，エンクレイヴと呼ばれる保護領域でプログラムを安全に実行するための CPU の機構である．エンクレイヴで IDS を実行することにより，クラウド内でも IDS の改ざんや IDS からの情報漏洩を防ぐことができる．エンクレイヴは通常のアプリケーションの一部であるため，高度な IDS の開発が行いやすく，システム全体の性能に及ぼす影響も小さい．ただし，攻撃者が IDS の実行を停止することは防げないため，クラウド外部の信頼できるリモートホストからハートビートを送ることで IDS の正常な動作を確認する．

我々は SGX をサポートした Xen 4.7 を用いて SGmonitor を実装した．SGmonitor では，SGX 仮想化を有効にした IDS VM と呼ばれる VM 内で IDS を SGX アプリケーションとして動作させる．IDS がデータの取得を行う際には，信頼できるハイパーバイザ内でデータの暗号化を行い，エンクレイヴ内で復号することで情報漏洩を防ぐ．また IDS の開発に LLView [11] を用いることにより，透過的な OS データの取得を実現する．SGmonitor を用いて実験

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

を行い，エンクレイヴ内で実行した IDS が監視対象 VM 内の OS データを正しく取得できることを確認した．また，SGmonitor を用いて実行した IDS と従来手法を用いて実行した IDS のデータ取得時間を測定し，SGmonitor によって生じるオーバヘッドを調べた．

以下，2 章でクラウドにおける IDS オフロードとその問題点について述べる．3 章で Intel SGX を用いて IDS を保護することにより，VM を安全に監視できるようにするシステム SGmonitor を提案する．4 章で SGmonitor の実装について述べる．5 章で SGmonitor の性能を調べ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 を安全に実行する手法として，VM を用いた IDS オフロードが提案されている [1]．この手法は図 1 のように，IDS を監視対象 VM の外側で実行し，VM 内部のシステムの監視を行う手法である．IDS オフロードを用いることにより，監視対象 VM に侵入されたとしても VM 内で IDS は動作していないため，IDS が無効化される危険はない．オフロードした IDS は監視対象 VM のメモリを解析して OS が管理しているデータを取得する．それにより，監視対象 VM 内で動作させた場合と同様にシステムの監視を行い，攻撃を検知することができる．例えば，VM 内で実行されているプロセスの一覧を取得することにより，不正なプログラムが実行されていないかをチェックすることができる．

しかし，IDS オフロードを行ったとしても，まだ IDS が攻撃を受ける可能性がある．オフロードした IDS はクラウド管理者の管理下におかれるが，クラウドの管理者は必ずしも信頼できるとは限らない．実際に，Google の管理者がユーザの機密情報を閲覧し，プライバシを侵害した事件が発生している [12]．また，サイバー犯罪の 28%は内部犯行であるという調査結果も出ている [13]．加えて，管理者の 35%は機密情報をのぞき見したことがあるという調査結果もある [14]．そのため，悪意のある管理者によってオフロードした IDS への攻撃が行われる危険性がある．また，クラウド外部の攻撃者がクラウドに侵入してオフロードした IDS への攻撃を行う危険性もある．その結果，オフロードした IDS が無力化させられたり，IDS が取得した VM 内の機密情報を盗まれたりする恐れがある．

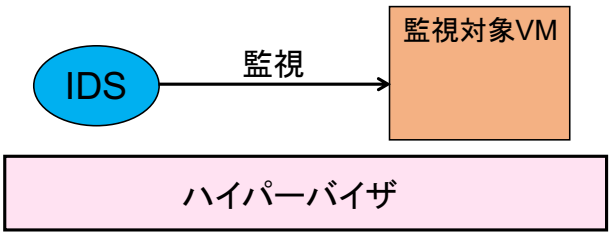

図 1　IDS オフロード

これらの問題を解決するために，VM の下で動作するハイパーバイザを信頼して，オフロードした IDS を安全に実行する手法が提案されてきた．例えば，ハイパーバイザ内に IDS をオフロードし，VM の監視を行うシステム BVMD [2] が提案されている．しかし，ハイパーバイザ内で高度な IDS を動作させるのは難しい．また，Self-Service Cloud （SSC）[3] を用いることで，サービスドメインと呼ばれる VM の中で IDS を安全に実行することもできる．サービスドメインにはユーザしかアクセスできないが，その中では通常のシステムが動作しているため，システムに脆弱性があった場合，IDS が攻撃を受ける危険性がある．Copilot [4] や HyperCheck [5]，RemoteTrans [6] はクラウド外部の信頼できるホストで IDS を実行し，監視を行うことができる．ただし，これらの手法ではクラウド内で IDS を実行できず，クラウドとは別のホストを用意する必要がある．

クラウド内のハイパーバイザを信頼せず，仮想化システムの外側で IDS を安全に動作させる手法も提案されている．HyperGuard [7] と HyperSentry [8] は CPU のシステムマネジメントモード（SMM）を用いて IDS を安全に実行する．また，Flicker [9] は AMD SVM や Intel TXT を用いて IDS を安全に実行することができる．しかし，SMM や SVM/TXT を用いて IDS を実行する場合，実行中は仮想化システムが停止してしまう．V-Met [10] はネストした仮想化を用いて，仮想化システムを VM 内で動作させ，その外側で IDS を安全に実行する．しかし，ネストした仮想化のオーバヘッドのため，仮想化システムの性能が低下するという問題がある．

3. SGmonitor

本稿では，Intel SGX を用いてクラウド内の IDS を保護することにより，VM を安全に監視できるようにするシステム SGmonitor を提案する．Intel SGX は，エンクレイヴと呼ばれる保護領域を用いてプログラムの安全な実行を保証する CPU の機構である．SGmonitor のシステム構成を図 2 に示す．SGmonitor では，IDS は SGX アプリケーションとして作成され，従来の IDS オフロードと同様に監視対象 VM が動作しているハイパーバイザ上で実行される．SGX アプリケーションはエンクレイヴと SGmonitor ライブラリで構成され，エンクレイヴ内の IDS はライブラリを介してハイパーバイザとの通信を行い，VM のメモリ上の OS データを取得する．IDS の動作を確認するために，クラウド外部の信頼できるリモートホストからハートビートを送る．このように，クラウド外部ではハートビートのためのホストだけを動作させればよい．

SGmonitor では以下のような脅威モデルを考える．ま

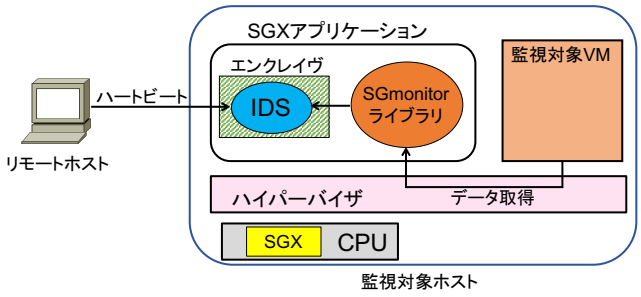

図 2　SGmonitor のシステム構成

ず，クラウドプロバイダは信頼し，プロバイダが提供しているクラウド内のハードウェアも信頼できるものとする．クラウドプロバイダは一度信用を失うと致命的であるため，この仮定は妥当であると考えられ，様々な研究で広く用いられている [15] [16] [17] [3] [6] [10]．また，クラウド内のハイパーバイザも信頼する．ハイパーバイザが信頼できる状態であることは様々な手法で確認できる．例えば，TPM を用いたリモートアテステーションによりハイパーバイザの正常起動を確認することができる．また SMM などのハードウェア機構を用いることにより，実行中にハイパーバイザの改ざんを検出することもできる [7] [5] [8] [9]．一方で，オフロードした IDS を動作させる OS などの実行環境や SGX アプリケーション内の SGmonitor ライブラリは信頼しない．本稿では，クラウド内の信頼できない管理者や外部の攻撃者が IDS への攻撃を行う状況を想定する．また，悪意のある管理者が作成した IDS によって，VM 内の機密情報が盗まれる攻撃も想定に入れる．

エンクレイヴ内で IDS を実行することにより，クラウド内の IDS を以下のように安全に実行することができる．エンクレイヴで IDS の実行を開始する際には SGX によってコードの電子署名が検査されるため，攻撃者が改ざんした IDS を実行することはできない．悪意ある管理者によって正しく署名された IDS が実行された場合でも，リモートアテステーションにより信頼できる第三者機関でチェックを行うことができる．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，実行中に IDS を改ざんすることもできない．それに加えて，エンクレイヴのメモリは暗号化されるため，IDS が取得した監視対象 VM 内の情報が漏洩することもない．ただし，攻撃者が IDS の実行を停止することは防げないため，クラウド外部のホストから定期的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．

エンクレイヴはアプリケーションの一部であるため，高度な IDS を比較的容易に開発することができる．SGmonitor では，エンクレイヴ内で動作する IDS はカーネルモジュールのように開発することが可能である．VM 内の OS データを取得する際にアドレス変換を行う必要があるが，SGmonitor では LLView [11] と呼ばれるツールを用いて透過的なアドレス変換を行う．これにより，IDS の開発者はオフロードを意識することなく IDS を開発することができる．

SGX アプリーケーションは OS 上の一つのアプリケーションであるため，仮想化システムの性能に影響を与えることはない．SMM や SVM/TXT のように，IDS の実行中に仮想化システムを停止させる必要はなく，VM を実行したまま監視を行うことができる．また，ネストした仮想化を用いる場合のように，仮想化システム全体に常にオーバヘッドがかかることもない．

4. 実装

我々は SGX をサポートした Xen 4.7 を用いて SGmonitor を実装した．

4.1 Intel SGX

SGX を用いることにより，アプリケーションはメモリ上に保護領域であるエンクレイヴを作成することができる．エンクレイヴのメモリにはエンクレイヴ内のコードだけがアクセスでき，エンクレイヴで実行されるコードはロード時に電子署名が検査される．SGX アプリケーションは，信頼できるエンクレイヴとエンクレイヴの外で動作する信頼できないコードで構成される．SGX には信頼できないコードからエンクレイヴを安全に呼び出す ECALL と，エンクレイヴから信頼できないコードを呼び出す OCALL と呼ばれる機構が用意されている．SGX の SDK [18] には ECALL と OCALL のインターフェースを定義するための EDL と呼ばれる言語が用意されている．

4.2 Xen-SGX

SGX を使用するにはエンクレイヴページキャッシュ（EPC）と呼ばれるエンクレイヴのためのセキュアな記憶域が必要になる．EPC はハードウェアによって提供される資源であり，我々の環境ではそのサイズは 93MB であった．通常，EPC は BIOS によって予約され，SGX ドライバをインストールした OS によって管理される．しかし，VM には EPC を割り当てることができず，SGX を使用することができない．

そこで，我々は仮想化環境に SGX 仮想化をサポートした Xen（Xen-SGX）[19] を用いた．Xen-SGX は Intel が研究用に提供している Xen のパッチであり，ハイパーバイザに EPC を管理する機能を追加することにより，完全仮想化 VM 内で SGX を使用することを可能にする．Xen-SGX では VM を立ち上げる際に，その VM に割り当てる EPC のサイズを決めることができる．SGmonitor では，IDS を実行するための VM として，EPC を割り当てた VM を作成し，その VM を IDS VM と呼ぶ．

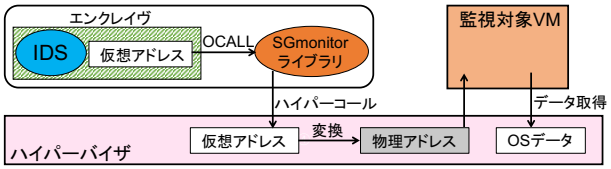

図 3　VM 内の OS データ取得

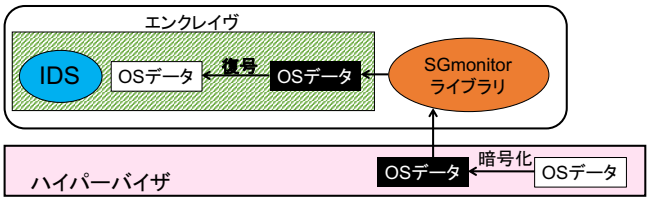

図 4　OS データの暗号化

4.3　OS データの取得

エンクレイヴ内の IDS は監視対象 VM のメモリ上の OS データを取得するために，図 3 のように，まず OCALL を用いてエンクレイヴ外部の SGmonitor ライブラリを呼び出す．これはエンクレイヴが直接ハイパーバイザを呼び出すことができないためである．そして，SGmonitor ライブラリはハイパーコールを用いてハイパーバイザを呼び出す．ハイパーバイザは仮想 CPU の CR3 レジスタが指している VM 内のページテーブルを参照して，取得しようとしているデータの仮想アドレスを対応する物理アドレスに変換する．その物理アドレスを用いて VM のメモリ上にある OS データをページ単位で取得する．

OS データをエンクレイヴに返す際に，SGmonitor では図 4 のように，取得した OS データをハイパーバイザ内で暗号化する．これにより，信頼できない SGmonitor ライブラリでの情報漏洩を防ぐ．暗号化された OS データが SGmonitor ライブラリを介してエンクレイヴに返されると，IDS は OS データを復号して監視を行う．データの暗号化・復号化を行うために，エンクレイヴとハイパーバイザに wolfSSL [20] の AES を移植した．

SGmonitor では取得した OS データをハッシュ表を用いてキャッシュする．これにより，IDS が同じデータを必要とした時には再度データを取得する必要がなくなる．また，OS データはページ単位で取得するため，同一ページ上の別のデータにアクセスする際にもハイパーコールを呼び出さずにアクセスすることができる．OS データをキャッシュすることにより最新のデータが得られない可能性があるが，データ取得のオーバヘッドを減らすことができる．

4.4　IDS の開発

SGmonitor では，Linux カーネルのヘッダファイルを用いて IDS を開発することができる．例えば，監視対象 VM 内で動作しているプロセス一覧をプロセスリストをたどって取得する IDS は図 5 のように記述される．この IDS は，

```
#include <linux/sched.h>
#include "addr.h"

void plist(void)
{
  struct task_struct *p;
  struct task_struct *init_task;
  char str[256];

  init_task = (struct task_struct *)0xffffffff81e11500;
  p = init_task;

  do {
    snprintf(str, 256, "pid: %d\n",p->pid);
    ocall_print(str);
    snprintf(str, 256, "name: %s\n", kstr(p->comm));
    ocall_print(str);

    p = list_entry(p->tasks.next,
                   struct task_struct,
                   tasks);
  } while (p != init_task);
}
```

図 5　プロセス一覧を取得する IDS

init_task 変数を開始点とする task_struct 構造体の循環リストからプロセス一覧を取得している．そして，OCALL を用いて SGmonitor ライブラリの ocall_print 関数を呼び出して，取得したプロセスの ID と名前の表示も行っている．現在の実装では，init_task 変数の仮想アドレスとして System.map の中の値を直接利用している．

LLView [11] を用いてプログラム変換を行うことにより，IDS は明示的に OCALL を呼び出すことなく，透過的に VM 内の OS データを取得することができる．LLView は IDS をコンパイルする際に生成された LLVM の中間表現を変換することにより，IDS が OS データにアクセスしようとした時に VM 内の OS データを取得するようにプログラムを変換する．LLVM の中間表現では，メモリからデータを読み込む際に load 命令を使用する．LLView を用いることにより，この load 命令の直前で OS データを取得する関数を呼び出し，取得したデータが置かれているエンクレイヴのメモリに対して load 命令を実行するように命令を置き換える．

例として，IDS をコンパイルして次のような中間表現が得られた場合について考える．

```
%1 = load i64, i64* %jiffies
%2 = udiv i64 %1, 250
```

この中間表現では OS 内の 64 ビットのグローバル変数 jiffies の値をローカル変数%1 に読み込み，250 で割って%2 に格納している．LLView を用いた場合，この中間表現は次のように変換される．

```
%1 = bitcast i64* %jiffies to i8*
```

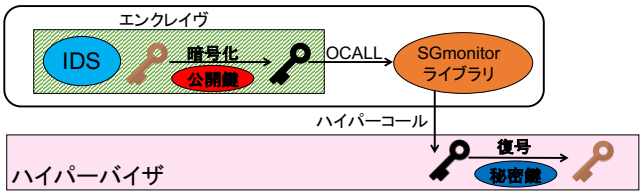

図 6　暗号鍵の共有

```
%2 = call i8* @g_map(i8* %1)
%3 = bitcast i8* %2 to i64*
%4 = load i64, i64* %3
%5 = udiv i64 %4, 250
```

この中間表現では jiffies のアドレスを 8 ビット整数のポインタにキャストして%1 に格納し，それを引数として g_map 関数を呼び出す．g_map 関数では OCALL を用いて SGmonitor ライブラリの関数を呼び出し，監視対象 VM から暗号化された OS データを取得して，そのデータを復号する．g_map 関数から返されたアドレスを元の 64 ビット整数のポインタにキャストし，そのアドレスにあるデータを%4 にロードする．

4.5　暗号鍵の共有

SGmonitor では公開鍵暗号を用いて，エンクレイヴとハイパーバイザ間で OS データを暗号化・復号化するための暗号鍵を共有する．図 6 のように，まずエンクレイヴ内の IDS が暗号鍵を生成し，それをハイパーバイザの公開鍵で暗号化する．ハイパーバイザの公開鍵はあらかじめ，IDS に埋め込んでおく．IDS は SGmonitor ライブラリを介して暗号化された暗号鍵をハイパーバイザに渡し，ハイパーバイザは自身の秘密鍵を用いて暗号鍵を復号する．秘密鍵はハイパーバイザだけが知っているため，ハイパーバイザ以外は暗号鍵を復号することができない．

暗号鍵を登録する際に，IDS はクラウド外部の信頼できる第三者機関と通信して暗号化された暗号鍵の電子署名を取得する．そして，暗号鍵と一緒にその電子署名をハイパーバイザに渡し，ハイパーバイザにおいて検証を行う．第三者機関は電子署名を行う際に，リモートアテステーションを通して署名を行う IDS が正しい IDS であることを確認する．正しい IDS に対してのみ電子署名を行うようにすることにより，正しい IDS だけが暗号鍵を登録することができる．現在のところ，電子署名の検証については未実装である．

また，IDS が正常に動作していることを確認するためのハートビートを送るために，IDS とリモートホスト間でも暗号鍵を共有する．その場合も上記と同様の方法で暗号鍵の登録を行う．

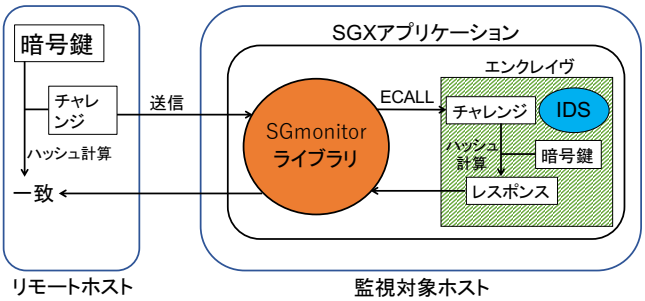

図 7　ハートビート

4.6　ハートビート

SGmonitor ではチャレンジ・レスポンス方式を用いて，リモートホストから IDS にハートビートを送信する．図 7 のように，まずチャレンジとして乱数をリモートホストから SGmonitor ライブラリに送信し，ECALL を用いてエンクレイヴ内の関数を呼び出し，チャレンジをエンクレイヴ内の IDS に渡す．IDS はリモートホストと共有している暗号鍵と受信したチャレンジからハッシュ値を計算した後，それをレスポンスとして SGmonitor ライブラリ経由でリモートホストに返す．リモートホストでも同様に暗号鍵とチャレンジからハッシュ値を計算し，それがレスポンスと一致すれば IDS の正常な動作を確認できる．ハッシュ値計算に暗号鍵が含まれるため，正しい IDS 以外は正しいレスポンスを返すことができない．現在のところ，ハートビートは未実装である．

5.　実験

SGmonitor を用いて VM 内のプロセス情報を取得する IDS が正常に動作するかを確認し，その性能を調べる実験を行った．実験には，Intel Xeon E3-1225 v5 の CPU，8GB のメモリを搭載したマシンを使用し，仮想化システムには Xen-SGX 4.7 を使用した．監視対象 VM には 2 個の仮想 CPU と 2GB のメモリを割り当て，OS には Linux 4.4 を使用した．IDS VM には 2 個の仮想 CPU と 2GB のメモリを割り当て，32MB の EPC を割り当てた．また，OS には Linux 4.13 を使用した．

5.1　IDS の動作確認

SGmonitor を用いて実装した図 5 のプロセス情報を取得する IDS が正常に動作することを確認した．動作確認のために，OCALL で SGmonitor ライブラリの関数を呼び出して，取得したプロセスの ID と名前を表示した．実行結果の一部を図 8 に示す．この実行結果から IDS が監視対象 VM 内で動作しているプロセスの情報を，プロセス ID が小さいものから順に取得できていることを確認できた．

5.2　プロセス情報の取得時間

IDS がプロセス情報を取得するのにかかる時間を測定し

```
pid: 0
name: swapper/0
pid: 1
name: systemd
pid: 2
name: kthreadd
pid: 3
name: ksoftirqd/0
pid: 4
name: kworker/0:0
pid: 5
name: kworker/0:0H
pid: 7
name: rcu_sched
```

図 8　IDS の実行の確認

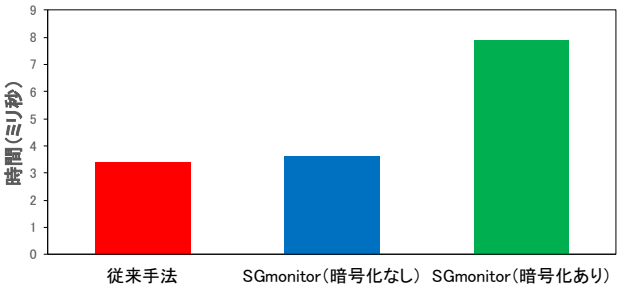

図 9　プロセス情報の取得時間

た．本実験では，IDS はプロセスリストを順番にたどるだけで，プロセス情報は表示しないようにした．比較として，OS データの暗号化を行う場合と行わない場合についてそれぞれ測定を行った．また，従来手法を用いてドメイン 0 にオフロードした IDS を実行した場合の取得時間も測定した．従来手法では，VM のメモリページを IDS のアドレス空間にマップしてプロセス情報を取得した．実験に用いた監視対象 VM では 166 個のプロセスが動作していた．VM 内のプロセス情報の取得にかかる時間を 10 回測定した時の平均値を図 9 に示す．

データの暗号化を行わない場合，SGmonitor における取得時間は従来手法と比較して 8%増加した．これは OCALL 呼び出しのオーバヘッドや VM のメモリを IDS にコピーするオーバヘッドが原因と考えられる．一方，データの暗号化を行う場合，暗号化を行わない場合と比べて 2.3 倍の時間がかかった．このオーバヘッドは CPU の AES 支援機構である AES-NI を用いることによって削減できると考えられる．

5.3　プロセス数と EPC サイズの影響

監視対象 VM のプロセス数を増やしながら，プロセス情報の取得にかかる時間がどのように増加するかを測定した．また，IDS VM に割り当てる EPC のサイズの影響を調べるために，EPC を 1MB だけ割り当てた場合についても測定を行った．いずれの場合も，取得するデータの暗号化を行った．比較として，従来手法を用いてドメイン 0 に

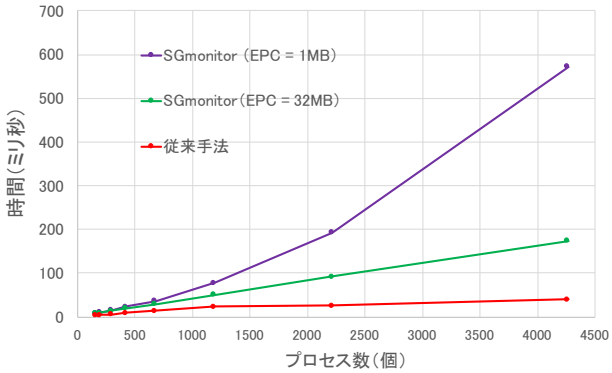

図 10　プロセス数と EPC サイズの影響

オフロードした IDS を実行した場合についても測定を行った．実験結果を図 10 に示す．

従来手法と EPC が 32MB の場合は，プロセス数に比例して取得時間が増加している．しかし，EPC が 1MB の場合はプロセス数の増加に伴い，取得時間の増加率が大きくなっている．加えて，EPC が 32MB の場合と比較して，取得に時間がかかっている．これは EPC のサイズが不足しており，EPC 上のデータが暗号化されて通常の DRAM に書き出されたことが原因であると考えられる．しかし，監視対象 VM 内で動作するプロセス数が一般的な数百個程度の場合には，EPC が 1MB しかなくとも 32MB の場合と比較して大きな差はない．

6.　関連研究

BVMD [2] は VM の下で動作するハイパーバイザを信頼して，ハイパーバイザ内に IDS をオフロードし，安全に VM の監視を行うシステムである．しかし，ハイパーバイザ内で高度な IDS を動作させるのは難しい．また，ハイパーバイザに埋め込むため IDS の更新などの管理が難しいという問題もある．SGmonitor では SGX アプリケーションとして IDS を作成するため，高度な IDS の開発も比較的容易であり，IDS の管理もしやすい．

Self-Service Cloud（SSC）[3] はユーザにだけ自身の VM を管理する権限を与えることができる．信頼できるハイパーバイザを用いることで，ユーザはサービスドメインと呼ばれるユーザだけがアクセスできる VM を作成することができる．そして，その VM 内でオフロードした IDS を安全に実行することができる．しかし，サービスドメイン内では OS も動作するため，そのシステムに脆弱性があった場合，IDS が攻撃を受ける可能性がある．SGmonitor ではエンクレイヴ内では IDS しか動作しないため，外部から脆弱性を利用した攻撃を行うのはより難しい．

RemoteTrans [6] はクラウドの外に信頼できるリモートホストを用意し，そのホスト内で IDS を安全に実行するシステムである．クラウド内のハイパーバイザを信頼し，リモートホストはハイパーバイザと暗号通信を行うことで

VM の内部情報を取得し，安全な監視を行うことができる．しかし，クラウドの資源を利用して IDS を実行できないという問題がある．SGmonitor では IDS はクラウド内で安全に実行することができる．

ハードウェア機構を用いることで IDS を安全に実行する手法も提案されている．Copilot [4] は専用の PCI カードを用意してメモリの内容をリモートホストに送信し，IDS を実行する． しかし，専用ハードウェアを用いるのはコストが高い．HyperGuard [7] は CPU のシステムマネジメントモード（SMM）を用いることで安全にハイパーバイザを監視することができ，HyperCheck [5] は SMM を用いてメモリの内容をリモートホストに送って安全に監視を行うことができる．HyperSentry [8] は SMM を用いることでハイパーバイザ内で安全に監視プログラムを実行することができる．しかし，SMM によるプログラムの実行は低速であり，IDS の性能が低下する．また，SMM でプログラムを実行する際にはシステム全体を停止させる必要がある．AMD SVM や Intel TXT を用いて IDS を安全に実行する Flicker [9] も提案されているが，IDS を実行する際には他の CPU コアを停止させる必要がある．SGmonitor では IDS 実行時に他の CPU コアを停止させる必要はない．

V-Met [10] はネストした仮想化を用いて仮想化システムをクラウド VM 内で動作させ，その外側で安全に IDS を実行する．V-Met では，IDS はクラウド VM の中の監視対象 VM のメモリを特定して必要な OS データを取得する．しかし，ネストした仮想化を用いることにより仮想化システムのオーバヘッドが大きくなるという問題がある．SGmonitor では監視対象 VM を含む仮想化システムは従来通りの性能で動作する．

7. まとめ

本稿では，Intel SGX を用いてクラウド内の IDS を保護し，正しい IDS だけが VM 内の情報を取得できるシステム SGmonitor を提案した．SGmonitor はエンクレイヴ内で IDS を動作させることによって IDS の改ざんを防ぎ，監視対象 VM から取得した機密情報の漏洩を防ぐことを可能にする．また，リモートホストからハートビートを送ることで IDS の無効化を検出することができる．我々は SGmonitor を Xen-SGX 4.7 に実装した．実験の結果，エンクレイヴ内の IDS から監視対象 VM 内の OS データを取得することが可能であることが分かった．また，データを暗号化して取得する際には，現在の実装ではデータの暗号化・復号化によるオーバヘッドが大きいことも分かった．

今後の課題は，安全な鍵共有の実装を完成させることや，リモートホストからのハートビートを実装することである．また，SGmonitor のセキュリティを向上させるためにリモートアテステーションを用いたエンクレイヴの識別や，ハイパーコールを呼ぶ際に渡す仮想アドレスの暗号化なども行う必要がある．加えて，AES-NI を使用することにより暗号化・復号化のオーバヘッドを削減するなど，SGmonitor の性能向上を目指す．さらに，VM のメモリだけでなく，ディスクやネットワークを監視する様々な IDS を実装することも今後の課題である．

参考文献

[1] Garfinkel, T. and Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, *Proceedings of Network and Distributed Systems SecuritySymposium*, pp. 191-206 (2003).
[2] Oyama, Y., Giang, T., Chubachi, Y., Shinagawa, T. and Kato, K.: Detecting Malware Signatures in a Thin Hypervisor, *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC' 12)*, pp. 1807- 1814 (2012).
[3] Butt, S., Lagar-Cavilla, H. A., Srivastava, A. and Ganapathy, V.: Self-service Cloud Computing, *Processings of Conference on Computer and Communications Security*, pp. 253-264 (2012).
[4] N. Petroni, Jr., T. Fraser, J. Molina, and W. Arbaugh: Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor, *Proceedings of the 13th Conference on USENIX Security Symposium*, pp.13-13 (2004).
[5] Wang, J., Stavrou, A. and Ghosh, A.: HyperCheck: A hardware-assisted Integrity Monitor, *Proceedings of International Symposium on Recent Advances in Intrusion Detection*, pp. 158-177 (2010).
[6] 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).
[7] Rutkowska, J., Wojtczuk, R. and Tereshkin, A.: HyperGuard, *Xen 0wning Trilogy, Black Hat USA* (2008).
[8] Azab, A. M., Ning, P., Wang, Z., Jiang, X., Zhang, X. and Skalsky, N. C.: HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity, *Proceedings of Conference on Computer and Communications Security*, pp. 38-49 (2010).
[9] McCune, J. M., Parno, B., Perrig, A., Reiter, M. K. and Isozaki, H.: Flicker: An Execution Infrastructure for TCB Minimization, *Proceedings of European Conference of Computer Systems*, pp. 315-328 (2008).
[10] Miyama, S. Kourai, K.: Secure IDS Offloading with Nested Virtualization and Deep VM Introspection, *Proceedings of the 22nd European Symposium on Research in Computer Security, part II*, pp.305-323 (2017).
[11] 尾崎雄一，山本裕明，光来健一：GPU を用いた OS レベルでの障害検知, 第 142 回 OS 研究会 (2018).
[12] TechSpot News: Google Fired Employees for Breaching User Privacy. http://www.techspot.com/news/40280-google-fired-employees-for-breaching-user-privacy.html (2010).
[13] PwC: US Cybercrime: Rising Risks, Reduced Readiness (2014).
[14] CyberArk Software: Global IT Security Service (2009).
[15] Li, C., Raghunathan, 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).
[16] Li, C., Raghunathan, A. and Jha, N. K.: Secure Virtual Machine Execution under an Untrusted Manage-

ment OS, *Proceedings on International Conference on Cloud Computing*, pp. 172-179 (2010).
[17] Santos, N., Gummadi, K. P. and Rodrigues, R.: Towards Trusted Cloud Computing, *Proceedings of Workshop on Hot Topics in Cloud Computing* (2009).
[18] Intel Corp., Intel Software Guard Extensions SDK, https://software.intel.com/en-us/sgx-sdk/
[19] Intel Corp., Xen SGX Virtualization Support, https://github.com/intel/xen-sgx/
[20] wolfSSL Inc.: wolfSSL — Embedded SSL Library for Applications, Devices, IoT, and the Cloud, http://www.wolfssl.com/