Intel SGXとSMMの組み合わせによるIDSの安全な実行機構

古賀　吉道[1]　光来　健一[1]

概要：インターネットに接続されたシステムへの攻撃を検知するために，侵入検知システム（IDS）が用いられている．しかし，システムの状態を監視して異常を検知するホストベースIDSは監視対象ホスト上で動作するため，安全に実行するのは容易ではない．例えば，システムが攻撃を受けた後にはそのシステムから正しい情報を取得できるとは限らない．また，IDSが改ざんされると無力化されてしまい，それ以降の攻撃を検知できなくなる．これまでに汎用CPUの機能を用いてIDSを安全に実行する手法が提案されてきたが，安全性や性能などの面で問題があった．本稿では，Intel CPUの機能であるSGXとシステムマネジメントモード（SMM）を組み合わせることで，安全にIDSを実行することが可能なシステムSSdetectorを提案する．SSdetectorはSGXのエンクレイヴ内でIDSを安全に実行し，SMMプログラムを用いてシステムのメモリデータの安全な取得を行う．エンクレイヴとSMMプログラム間でメモリデータを暗号化することで，取得したメモリデータからの情報漏洩を防ぐ．我々はSGX仮想化をサポートしたKVMを用いてVMのUEFI BIOSを変更することでSSdetectorを実装し，IDSによるOSデータの取得時間を調べた．

1. はじめに

近年，インターネットに接続されたシステムへの攻撃が数多く報告されている．攻撃の糸口となるシステムの脆弱性を完全に取り除くのは困難であるため，侵入検知システム（IDS）を用いてシステムを監視し，システムが攻撃を受けた場合には管理者に報告する必要がある．しかし，システムの状態を監視して異常を検知するホストベースIDSは監視対象ホスト上で動作するため，安全に実行するのは容易ではない．例えば，システムが攻撃を受けた後はIDSがそのシステムから正しい情報を取得できるとは限らない．また，IDSが改ざんされると無力化されてしまい，それ以降の攻撃を検知できなくなる．

これまでに汎用CPUの機能を用いてIDSを安全に実行する手法が提案されてきたが，安全性や性能の面で問題があった．例えば，IntelやAMDのCPUの動作モードの1つであるシステムマネジメントモード（SMM）を用いた手法 [1] がある．この手法は，IDSをBIOS内のSMMプログラムとして動作させることで監視対象システムから保護し，システムのメモリの安全な監視を可能にする．しかし，SMMでのプログラム実行は低速であり，実行中はシステムが停止するため，性能面での影響が大きい．一方，最近のIntel CPUに搭載されたセキュリティ機能であるSoftware Guard Extensions (SGX) を用いる手法 [2] もある．この手法はSGXのエンクレイヴ内でIDSを安全に動作させ，ハイパーバイザ経由でVMのメモリの監視を可能にする．しかし，ハイパーバイザに脆弱性がある場合には情報が漏洩する恐れがある．

この問題を解決するために，本稿ではSGXとSMMを組み合わせることで，安全にIDSを実行することが可能なシステムSSdetectorを提案する．SSdetectorはSGXのエンクレイヴ内でIDSを実行し，SMMで動作するプログラムがシステムのメモリデータの取得のみを行う．それにより，SMMによる実行速度の低下を最小限に抑えつつ，安全にIDSを実行する．SSdetectorはエンクレイヴとSMMプログラム間でメモリデータを暗号化することで，取得したメモリデータからの情報漏洩を防ぐ．システム管理者は定期的にIDSと通信することにより，IDSの正常動作や検知結果の確認を行う．

我々はSSdetectorのSMMプログラムをオープンソースのUEFI BIOSであるTianoCore [3] に実装した．このSMMプログラムはIDSから受け取ったOSの仮想アドレスを物理アドレスに変換し，そのアドレスのメモリデータをIDSに返す．現在のところ，SGX仮想化をサポートしたKVMを用いてVMのBIOSを変更することにより

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

SSdetector を動作させている．SSdetector を用いて実験を行い，IDS が監視対象システムの OS データを取得できることを確認した．また，IDS からメモリデータを取得するのにかかる時間を測定し，SMM プログラムの呼び出しやデータの暗号化などによって生じるオーバヘッドを調べた．

以下，2 章で IDS の従来の安全な実行手法について述べる．3 章で Intel SGX と SMM を組み合わせることで，IDS を安全に実行可能にするシステム SSdetector を提案する．4 章で SSdetector の実装について述べる．5 章で SSdetector 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. ホストベース IDS の安全な実行

2.1 IDS の安全な実行

IDS を安全に実行するための要件として，(1) 監視対象システムの機能を用いずに攻撃を検知できる，(2) 攻撃者に侵入されても IDS は改ざん・盗聴されない，(3) IDS が攻撃者によって停止されない，もしくは停止されたことを検知できる，の 3 点が挙げられる．監視対象システムの機能を用いてシステムの監視を行う場合，システムが攻撃を受けた後は IDS がそのシステムから正しい情報を取得できる保証はなくなる．IDS が改ざんされるとそれ以降の攻撃を正しく検知できなくなる．IDS が取得したシステム情報を盗聴されると，システムへの攻撃に成功しなくても IDS から機密情報の一部を盗むことができる．また，IDS は攻撃を受けずに動き続けることが望ましいが，少なくとも IDS が停止させられたことを検知できれば管理者は対処を行うことができる．

これまで，汎用 CPU の機能を用いてこれらの要件を満たす IDS が提案されてきたが，安全性や性能などの面で問題があった．

2.2 SMM を用いた IDS

SMM は Intel や AMD の CPU の動作モードの一つであり，BIOS によってのみ使用可能で，OS でさえアクセスができない独立した環境を提供する．SMM で動作するプログラムは SMRAM と呼ばれる専用メモリに置かれ，SMM でしかアクセスすることができない．SMM プログラムはシステムマネジメント割り込み（SMI）と呼ばれる割り込みを発生させることにより実行される．SMI はマシンの起動時に発生するほか，特定の I/O ポートに書き込むことなどによって発生させることができる．

IDS を SMM プログラムとして実行することで，安全な実行のための 3 つの要件を満たすことができる．例えば，HyperGuard [1] は SMM で動作する IDS がハイパーバイザのメモリを監視することにより改ざんを検知する．この IDS がシステムへの侵入者からの攻撃を受けることはない．しかし，SMM でのプログラム実行は低速であり，実行中はシステムが停止するという欠点がある．HyperCheck [4] は SMM でネットワークドライバを動かしてメモリデータをリモートホストに送信し，リモートホストで監視を行う．この手法ではリモートホストで動作する IDS の安全性が課題となる．

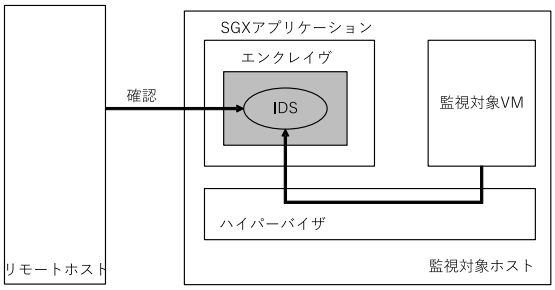

図 1 SGX を用いた IDS

2.3 Intel SGX を用いた IDS

SGX は第 6 世代以降の Intel の CPU に搭載されているセキュリティ機構である．SGX を用いることで，アプリケーションのメモリ上にエンクレイヴと呼ばれる保護領域を生成し，安全にプログラムを実行することができる．SGX によってプログラムの電子署名が検査されるため，攻撃者は改ざんしたプログラムを実行することができない．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ているため，攻撃者はエンクレイヴ内で実行中のプログラムを改ざんすることができない．加えて，エンクレイヴのメモリは暗号化されているため．エンクレイヴ内のデータが攻撃者によって盗聴されることもない．

IDS をエンクレイヴ内で動作させることにより，安全な実行の 3 つの要件を満たすことができる．例えば，SGmonitor [2] では図 1 のようにエンクレイヴ内の IDS が仮想マシン（VM）のメモリデータを取得することで監視を行う．システムへの侵入者は IDS の改ざんや盗聴を行うことはできない．ただし，攻撃者は IDS が動作する SGX アプリケーションを容易に停止できるため，リモートホストから IDS の正常動作を安全に確認する．エンクレイヴが VM のメモリデータを直接取得するのは難しいため，SGmonitor ではハイパーバイザを経由して VM のメモリを監視する．そのため，ハイパーバイザに脆弱性があるとメモリデータを改ざんされたり盗聴されたりする恐れがある．また，仮想化システムの監視しかできないという課題もある．

3. SSdetector

本稿では，SGX と SMM を組み合わせることにより，IDS を安全に実行できるようにするシステム SSdetector を提案する．

3.1 脅威モデル

SSdetector では以下のような脅威モデルを考える．監視対象システムの CPU と BIOS 内の SMM プログラムは信頼できるものとする．BIOS 内の SMM プログラムへの攻撃はハイパーバイザへの攻撃と比較してはるかに難しいため，ハイパーバイザを信頼するシステム [2] よりも IDS を安全に実行することができる．外部の攻撃者がネットワーク経由で攻撃を行うことを想定し，攻撃者が BIOS の入れ替えを行うことはできないことを仮定する．一方で，IDS を実行する OS などの実行環境や SGX アプリケーション内のエンクレイヴ以外のコードは信頼しない．

3.2 SGX と SMM を用いた IDS

SSdetector の IDS は SGX と SMM を用いて OS のメモリデータを安全に取得することによりシステムの監視を行う．SGX が提供する保護領域であるエンクレイヴ内で IDS を実行し，CPU によるメモリの暗号化および整合性検査により IDS の改ざんや盗聴を防ぐ．攻撃者による IDS の停止を検知するために，リモートホストから IDS に定期的にハートビートを送信する．一方，SMM で動作するプログラムがシステムのメモリデータの取得を行う．SMM は独立した実行環境を提供するため，攻撃を受けることなく安全にシステム全体のメモリデータを取得できる．メモリデータの取得のみを行うことで SMM による実行速度の低下やシステムの性能への影響を最小限に抑えることができる．

SSdetector のシステム構成を図 2 に示す．IDS を実行する SGX アプリケーションは監視対象システムの OS 上で動作し，エンクレイヴと SSdetector ランタイムからなる．ランタイムはエンクレイヴ内の IDS と SMM プログラムの間でデータの中継を行う．ランタイムは SGX によって保護されていないため，攻撃を受ける可能性がある．BIOS の中には SSdetector のためにメモリデータを取得する SMM プログラムがある．IDS がメモリデータを取得するためにハイパーバイザを必要としないため，システムが仮想化されている必要はない．IDS の正常動作や検知結果を確認するためにリモートホストが用いられる．

エンクレイヴ内の IDS がシステムを監視するために OS データを必要とした際には，SGX の機能を用いてエンクレイヴの外部で動作する SSdetector ランタイムを安全に呼び出す．ランタイムを呼び出すのは SGX の仕様上，エンクレイヴ内から直接 SMM プログラムを呼び出すことができないためである．呼び出したランタイムはソフトウェア割り込みを発生させることにより，SMM プログラムを呼び出す．その際に，取得しようとしている OS データの仮想アドレスを SMM プログラムに渡す．ランタイム経由で情報が漏洩するのを防ぐために，SSdetector はエンクレイヴと SMM プログラム間でやりとりするすべてのデータを暗号化する．

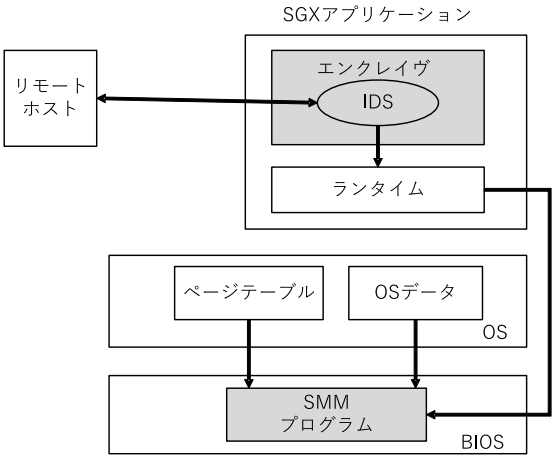

図 2 SSdetector のシステム構成

呼び出された SMM プログラムがシステムのメモリからデータを取得する際には，IDS から渡された仮想アドレスを物理アドレスに変換する．IDS には監視対象システムの OS データの仮想アドレスしか分からないのに対し，SMM プログラムは物理アドレスでしかメモリにアクセスすることができないためである．そこで，メモリ上の OS のページテーブルを探索することでアドレス変換を行う．SMM プログラムがメモリデータを取得すると SSdetector ランタイムに戻り，エンクレイヴ内の IDS にそのメモリデータが渡される．最終的に，リモートホストが IDS と通信することにより IDS の検知結果を取得する．

4. 実装

我々は SSdetector の SMM プログラムを EDK II [5] を用いてオープンソースの UEFI BIOS である TianoCore に実装した．また，IDS を動作させる SGX アプリケーションを Intel SGX SDK 2.13.3 [6] を用いて実装した．

実機の BIOS を変更するのは難しく，起動しなくなる恐れがあるため，現在のところ，図 3 のように SSdetector を VM 内で動作させている．SGX をサポートした VM を作成できるようにするために，SGX 仮想化のためのパッチを適用した KVM SGX [7] と QEMU SGX [8] を用いた．

4.1 IDS のための SGX アプリケーション

SSdetector では，SGX アプリケーションを起動するとまず SSdetector ランタイムが実行され，IDS のためのエンクレイヴが作成される．IDS の実行を開始する際には，図 4 のように ECALL と呼ばれる SGX の機構を用いてエンクレイヴ内部の IDS の関数を安全に呼び出す．その際に，ランタイムとエンクレイヴの間でメモリデータを共有するためのバッファのアドレスを IDS に渡す．一方，エンクレイヴ内の IDS は SSdetector ライブラリ経由でランタ

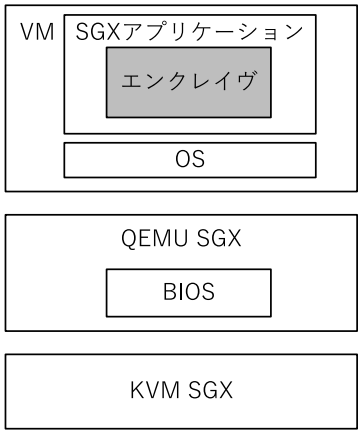

図 3 VM を用いた SSdetecctor の実装

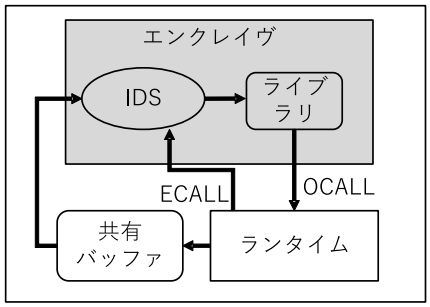

図 4 SGX アプリケーションの構成

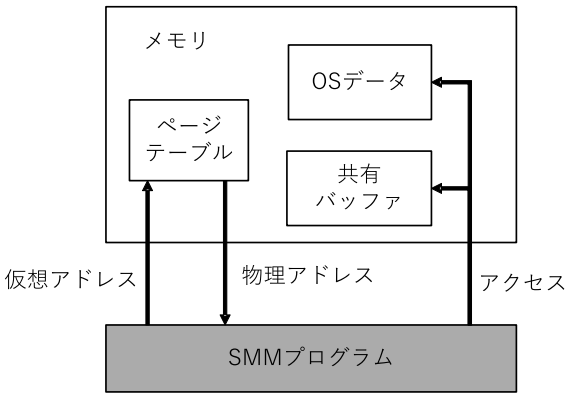

図 5 SMM プログラムにおけるアドレス交換

イムを呼び出す．その際に，OCALL と呼ばれる SGX の機構を用いてランタイムの関数を安全に呼び出す．これらの ECALL と OCALL のインタフェースは EDL ファイルを用いて定義した．

SSdetector ランタイムは，ioperm システムコールを用いて入出力許可を設定した上で 0xb2 番の I/O ポートに書き込みを行い，SMI を発生させる．SMM プログラムの中の SMI ハンドラに情報を渡すために，OS データの仮想アドレスを CPU の RDI レジスタに，共有バッファの仮想アドレスを RSI レジスタに格納する．SMI ハンドラから戻ってきた際には，共有バッファに要求したメモリデータが格納されている．OCALL の呼び出しからエンクレイヴ内の SSdetector ライブラリに戻る際にも，その共有バッファを介してメモリデータが渡される．ライブラリはそのメモリデータをコピーしてキャッシュすることにより，SMM プログラムの呼び出し回数を削減する．

SSdetector ライブラリと SMM プログラムの間でデータの暗号化と復号化を行うために，エンクレイヴ内で動作するライブラリに wolfSSL の AES 関数を移植した．IDS が SMM プログラムを呼び出す際には，SSdetector ライブラリにおいて OS データとバッファの仮想アドレスを暗号化する．それぞれの仮想アドレスは 8 バイトしかないため，２つの仮想アドレスを合わせた 16 バイトのデータに対して AES 関数を用いて暗号化を行う．この暗号化については現在のところ未実装である．他方，SMM プログラムにおいて暗号化されたメモリデータについては SSdetector ライブラリにおいて復号する．

4.2 IDS のための SMM プログラム

SMM プログラムが監視対象システムのメモリ全体にアクセスできるようにするために，SSdetector では従来の BIOS ではなく，その後継である UEFI BIOS を用いる．UEFI BIOS は 64 ビットモードで動作し，4GB を超えるメモリにもアクセス可能である．ただし，TianoCore においては SMM でアクセス可能なメモリ領域が制限されているため，SSdetector ではメモリ全体にアクセスできるようにこの制限を解除する．TianoCore は SMM のためのページテーブルを作成してメモリアクセスを制限しているため，SSdetector ではすべてのメモリにアクセスできるようにページテーブルを作成する．

SMI によって呼び出された SMM プログラムはまず，CPU の RDI レジスタと RSI レジスタに格納された仮想アドレスを復号する．次に，CPU の CR3 レジスタの値を取得して OS メモリ上のページテーブルを特定する．SMI ハンドラが呼び出された時点で CR3 レジスタは必ず SGX アプリケーションを実行しているプロセスのページテーブルを指している．そのページテーブルを用いて図 5 のように OS データの仮想アドレスを物理アドレスに変換し，共有バッファの仮想アドレスも物理アドレスに変換する．その後，OS のメモリデータを読み出し，暗号化して共有バッファに書き込む．データの暗号化・復号化については EDK II に含まれる暗号パッケージを利用した．

4.3 LLView の利用

SSdetector では LLView フレームワーク [9] を用いることで，Linux カーネルのヘッダファイルを利用して OS

```
koga@koga-Standard-PC-Q35-ICH9-2009:~/BannerEnclave$ sudo ./app
[sudo] koga のパスワード:
Linux version 5.8.0-36-generic (buildd@lgw01-amd64-027) (gcc (Ubuntu 9.3.0-17ubu
ntu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #40~20.04.1-Ubuntu SMP
 Wed Jan 6 10:15:55 UTC 2021 (Ubuntu 5.8.0-36.40~20.04.1-generic 5.8.18)
```

図 6 取得した OS のバージョン情報

表 1 比較対象

	SGX	SMM	暗号化
SSdetector	○	○	○
暗号化なし	○	○	×
SGX なし	×	○	○
SMM なし	○	×	○

データを監視する IDS を開発することができる．LLView は IDS をコンパイルして生成された LLVM の中間表現に対してプログラム変換を行い，IDS が透過的に監視対象システムのメモリにアクセスできるようにする．システムのメモリにアクセスする際に OCALL を呼び出すようにし，呼び出された SSdetector ランタイムにおいて SMI を発生させるように LLView を修正した．

5. 実験

SSdetector の有効性を調べるために，OS のバージョン情報を取得する IDS を用いて実験を行った．この IDS は Linux カーネルの linux_banner 変数の仮想アドレスを指定して，その中に格納されているバージョン文字列を取得する．実験に使用したマシンの CPU は Intel Core i7-9700 であり，メモリは 16GB であった．本実験では，SGX 仮想化をサポートした KVM SGX v5.6.0-rc5-r2 および QEMU SGX v4.0.0-r1 を用いて SSdetector を VM 内で動作させた．この VM には仮想 CPU を 1 個，メモリを 2GB 割り当て，VM 内では Linux 5.8.0-36-generic を動作させた．

5.1 IDS の動作確認

SSdetector を用いてエンクレイヴ内で IDS を実行し，SMM プログラム経由でシステムのメモリから取得した OS のバージョン情報を出力した．実行結果は図 6 のようになった．これより，監視対象システムのバージョン情報が正しく取得できていることが確認できた．

5.2 IDS の性能

SSdetector を用いて IDS を実行し，OS のバージョン情報の取得にかかる時間を測定した．比較として，(1) エンクレイヴと SMM プログラムの間で暗号化を行わない場合，(2) SGX を用いない場合，(3) SMM を用いない場合についても計測した．SGX を用いない場合は，エンクレイヴを用いずに IDS を実行し，SSdetector ライブラリが直接ランタイムの関数を呼び出すようにした．SMM を用いない場合には，SSdetector ランタイムは OS に組み込んだデバイスドライバを呼び出すことでシステムのメモリデータを取得するようにした．表 1 にそれぞれの条件で用いた機能の組み合わせを示す．

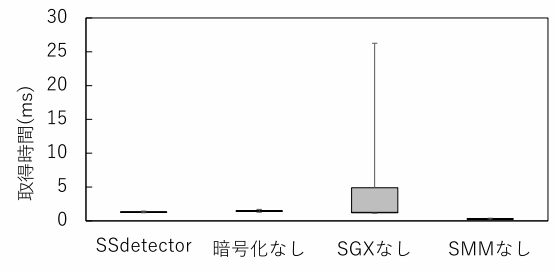

図 7 バージョン情報の取得時間

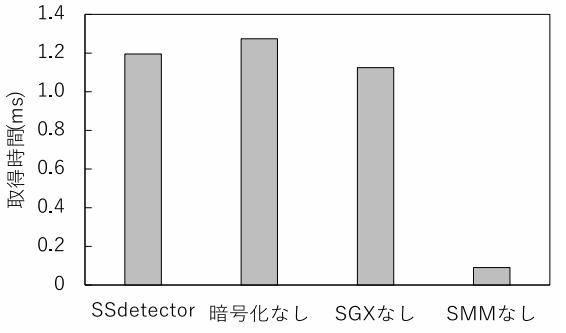

図 8 取得時間の最小値

IDS による OS のバージョン情報の取得時間を 50 回ずつ計測した結果を図 7 に示す．取得時間のばらつきが大きい場合があったため，箱ひげ図で最大値，最小値，四分位数を示している．SMM を用い，かつ SGX を用いない場合にばらつきが大きくなっており，特に最大値と第 3 四分位数が非常に大きくなっていることがわかる．これは SMI を発生させてから SMI ハンドラが呼び出されるまでの時間のばらつきが大きいことが原因だと考えられる．一方，SGX を用いた場合には SMM を用いてもばらつきはほぼなかった．このような結果になる原因は現在，調査中である．

図 7 ではばらつきが大きく比較が難しいため，図 8 に取得時間の最小値のみを示す．これより，SMM プログラムを呼び出すことによって $1.1ms$ のオーバヘッドが生じていることが分かった．これはバージョン情報の取得時間の 92.4 %を占める．しかし，SMM を用いない場合には IDS は安全にシステムのメモリデータを取得することができないため，安全性の確保と引き換えに必要となるオーバヘッドである．一方，SGX を用いることによって生じるオーバヘッドは $0.07ms$ であり，バージョン情報の取得時間全体の 5.9 %であった．このことから，SGX を用いるオーバヘッドは小さいことがわかる．

5.3 実機との性能比較

SMI を発生させて SMI ハンドラから戻ってくるだけの時間を 50 回ずつ測定した．実機の BIOS には変更を加えていない．この実行時間は図 9 のようになった．VM 内で

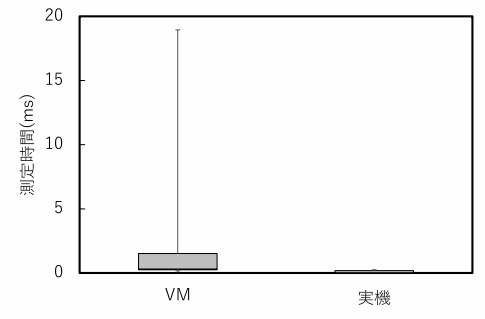

図 9 SMI の処理にかかる時間

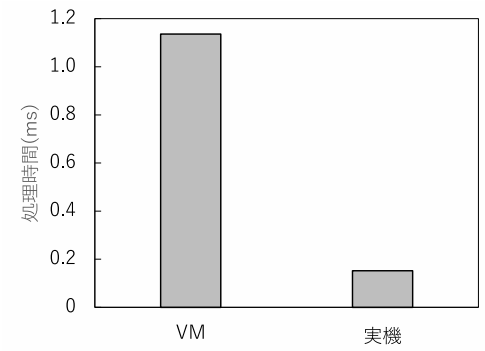

図 10 SMI 処理時間の最小値

SMI を発生させた場合のみばらつきが大きくなっていることがわかる．これは VM 内では SMI が仮想化されており，QEMU 内の BIOS を呼び出す時間がばらつくためだと考えられる．図 10 に測定時間の最小値を示す．実機での SMI の処理時間は VM を使用した場合に比べて 7.2 倍高速であることが分かった．SMI の処理時間は SSdetector によるバージョン取得時間の 99.6 %を占めているため，SSdetector を実機で動作させた場合には大幅なオーバヘッド削減が見込まれる．

6. 関連研究

SMM を用いたホストベース IDS として HyperGuard [1]，HyperCheck [4]，HyperSentry [10] がある．HyperGuard は SMM で IDS を実行するため，SMM による性能への影響が大きい．また，IDS を更新するたびに BIOS を変更する必要がある．一方，HyperCheck は SMM でメモリデータの転送のみを行い，リモートホストで監視を行う．そのため，SMM による性能低下を抑えることができる．しかし，リモートホストの安全性は保証されない．また，NIC ごとに SMM で動作するドライバを開発する必要がある．SSdetector ではエンクレイヴ内で安全に監視を行うことができ，SMM で動くプログラムはどの実行環境でも共通である．

HyperSentry は SMM でハイパーバイザへのエージェントの挿入のみを行う．その後で割り込みを禁止し，1 つを除くすべての CPU コアを停止させてエージェントを実行し，安全にシステムの監視を行う．そのため，時間のかかる監視を行うと，割り込み禁止時間やシステムの停止時間が長くなる．SSdetector では SMM でメモリデータを取得している間しかシステムは停止しない．

SGX を用いたホストベース IDS として SGmonitor [2] や SCwatcher [11] がある．SGmonitor はエンクレイヴ内で IDS を実行し，ハイパーバイザを呼び出すことで VM のメモリデータを監視する．また，エンクレイヴ内でファイルシステムを動作させることで VM の仮想ディスクを監視することもできる．しかし，対象が仮想化システムに限定され，ハイパーバイザを信頼する必要がある．SSdetector は SMM プログラムを利用することで，仮想化されていないシステムでも安全に監視を行うことができる．

SCwatcher は，SGX 向け実行環境である SCONE [12] を用いて既存の IDS をエンクレイヴ内で実行可能にする．さらに，エンクレイヴ内で proc ファイルシステムを提供することにより，監視対象 VM の情報を既存のインタフェースで取得することができる．このシステムと組み合わせることで，SSdetector でも既存の IDS を動かすことが可能になる．

SGX を用いたネットワークベース IDS には S-NFV [13] や SEC-IDS [14] がある．S-NFV は Snort の内部状態をエンクレイヴ内に格納することによって保護する．エンクレイヴ内に移動した内部状態はエンクレイヴ内のコードのみがアクセスすることができ，外部からの攻撃を受けない安全な API 経由で利用する．これにより，ネットワークフローごとの情報などを盗み見られないようにすることができる．

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 [15] を用いる．また，DPDK を用いることでエンクレイヴ内でネットワークパケットを効率よく取得し，通常の Snort とほぼ同等の性能を達成する．しかし，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．SSdetector では SMM を利用しているため，メモリデータを取得している間に攻撃者が書き換えることはできない．

IDS 用ではないが SGX と SMM を組み合わせたシステムとして Aurora [16] や Kshot [17] がある．Aurora はエンクレイヴが安全にデバイスを利用することを可能にする．エンクレイヴがクロックやネットワークなどのデバイスにアクセスする際には SMI を発生させて SMM プログラムを呼び出し，SMM でデバイスドライバを実行してデバイスにアクセスする．エンクレイヴと SMM プログラムは共有メモリを経由して暗号化されたデータをやり取りする．こ

れにより，エンクレイヴからデバイスまでの安全な経路が確保される．SSdetector では SMM プログラムを経由してシステムのメモリにアクセスし，監視を行う点が異なる．

KShot は OS やパッチシステムを信頼せずに，カーネルを動かしたままでパッチを適用することを可能にする．エンクレイヴを用いて安全にカーネルパッチをダウンロードして前処理を行い，予約されたメモリに書き込む．そして，SMM プログラムを呼び出し，SMM でカーネルにパッチを適用する．

7. まとめ

SGX と SMM を組み合わせることで，IDS をより安全に実行可能にするシステム SSdetector を提案した．SSdetector では，SGX のエンクレイヴ内で IDS を実行し，SMM プログラムではシステムのメモリデータの取得のみを行う．エンクレイヴと SMM プログラム間でメモリデータを暗号化することで，取得したメモリデータからの情報の漏洩を防ぐ．実験の結果，エンクレイヴ内から監視対象システムの OS データを取得できることが確認できた．また，データを取得する際は SMI を発生させ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，SMM プログラムが取得したメモリデータの整合性検査を行えるようにして取得中の改ざんを検知できるようにすることである．また，LLView の実装を完了させ，様々な OS データを取得する IDS を動かして性能評価を行うことも必要である．

謝辞

本研究成果は，国立研究開発法人情報通信研究機構の委託研究により得られたものです．

参考文献

[1] J. Rutkowska and R. Wojtczuk. Preventing and Detecting Xen Hypervisor Subversions. *Blackhat Briefings USA*, 2008.

[2] 中野智晴, 光来健一ほか. 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 (CSS), 2019.

[3] TianoCore Open Source Community. TianoCore. `https://www.tianocore.org/`. (Accessed on 06/24/2021).

[4] J. Wang, A. Stavrou, and A. Ghosh. HyperCheck: A Hardware-Assisted Integrity Monitor. In *International Workshop on Recent Advances in Intrusion Detection*, pp. 158–177. Springer, 2010.

[5] EDK II Project. EDK II. `https://github.com/tianocore/edk2`. (Accessed on 06/24/2021).

[6] Intel Corporation. Intel Software Guard Extensions SDK for Linux. `https://01.org/intel-software-guard-extensions/downloads`. (Accessed on 06/24/2021).

[7] Intel Corporation. KVM SGX. `https://github.com/intel/kvm-sgx`. (Accessed on 06/24/2021).

[8] Intel Corporation. QEMU SGX. `https://github.com/intel/qemu-sgx`. (Accessed on 06/24/2021).

[9] Y. Ozaki, S. Kanamoto, H. Yamamoto, and K. Kourai. Detecting System Failures with GPUs and LLVM. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*, pp. 47–53, 2019.

[10] A. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. Skalsky. HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity. In *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 38–49, 2010.

[11] 河村拓実, 光来健一. SGX 向け実行環境 SCONE を用いた VM の安全な監視機構. 研究報告システムソフトウェアとオペレーティング・システム (OS), Vol. 2020, No. 3, pp. 1–8, 2020.

[12] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, and M. Stillwell. SCONE: Secure linux Containers with Intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 689–703, 2016.

[13] M. Shih, M. Kumar, T. Kim, and A. Gavrilovska. S-NFV: Securing NFV States by Using SGX. In *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pp. 45–48, 2016.

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

[15] C. Tsai, D. Porter, and M. Vij. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pp. 645–658, 2017.

[16] H. Liang, M. Li, Y. Chen, L. Jiang, Z. Xie, and T. Yang. Establishing Trusted I/O Paths for SGX Client Systems with Aurora. *IEEE Transactions on Information Forensics and Security*, Vol. 15, pp. 1589–1600, 2019.

[17] L. Zhou, F. Zhang, J. Liao, Z. Ning, J. Xiao, K. Leach, W. Weimer, and G. Wang. KShot: Live Kernel Patching with SMM and SGX. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–13. IEEE, 2020.