

命令フィルタの導入による仮想デバイスエミュレーションの脆弱性回避

石黒 健太¹ 小笠原 純也¹ 河野 健二¹

概要: 仮想化を実現するハイパーバイザはクラウドサービスの基盤技術として用いられている。ハイパーバイザの脆弱性は致命的であるにもかかわらず、多くの脆弱性が報告されている。例えば、KVM の命令エミュレーションに権限昇格の脆弱性が存在することが知られている。この脆弱性の要因のひとつは、メモリマップド I/O (MMIO) 領域にアクセスする命令をすりかえることで、任意の命令のエミュレーションを行わせることができるためである。現在の CPU は互換性維持のために複雑な動作を行う命令があり、すべての動作を正しくエミュレートすることは難しい。本論文では不正な命令エミュレーションによる脆弱性を回避するために、MMIO 領域にアクセスする命令を限定する命令フィルタを導入する。MMIO 領域にアクセスする命令は限定的であり、互換性維持のために複雑な動作をするような命令を用いることはない。本論文で提案する命令フィルタでは、MMIO 領域へのアクセスを行うことのない命令をフィルタアウトすることで、任意の命令をエミュレートさせることができないようにする。提案方式を KVM (Linux Kernel 4.8) に実装し、権限昇格に必要な命令のすり替えを行う攻撃コードを実行したところ、すり替えられた命令のエミュレーションを拒否することを確認した。また、命令フィルタの導入によるオーバーヘッドは最大で 2.5% であった。

キーワード: ハイパーバイザ, エミュレーション, デバイス仮想化, セキュリティ

1. はじめに

仮想化はクラウドサービスを実現するための基盤技術の一つであり、Google App Engine [1] や Amazon EC2 [2] などに代表されるように広く活用されている。仮想環境上ではハイパーバイザによって物理ハードウェアリソースの抽象化を行うことで、単一の物理マシン上で複数の仮想マシンを並行に動かすことを可能にしている。ハイパーバイザの脆弱性はその上で動く複数の仮想マシンに影響を与えるため、一般的なソフトウェアの脆弱性と比べてより深刻な被害につながる場合が多い。

ハイパーバイザの一つである KVM [3] では 300 K lines of code (LOC) 以上のコード量 [4] となっていると共にその機能は複雑化している。そのため脆弱性の原因となるような不具合を事前にすべて排除することは現実的に不可能になっている。実際に、ハイパーバイザには多くの脆弱性があることが指摘されている。例えば、KVM では 104 個の共通脆弱性識別子が付与されている。また、Xen [5] では 205 個の脆弱性が Xen Security Advisories [6] (XSA)

にて報告されている。ハイパーバイザの中でもエミュレーションに関する部分には脆弱性が生まれやすく、XSA で報告されている Xen に直接関係する 144 個の脆弱性の内 13.19% がエミュレーションに関する脆弱性となっている [7]。

本論文ではハイパーバイザに固有の脆弱性である“エミュレートされる命令のすり替え”を回避する手法を提案する。エミュレートされる命令のすり替えによる脆弱性とは、仮想マシンがメモリマップド I/O (MMIO) 領域にアクセスする命令をすり替えることで、ハイパーバイザに任意の命令をエミュレートさせることができるという脆弱性である [8]。この脆弱性は、MMIO のエミュレーションというハイパーバイザ固有の機能に潜む脆弱性であるため、バッファオーバーフローや use-after-free などの典型的な脆弱性と比べて十分に対策が知られていない。

エミュレートされる命令のすり替えによる脆弱性では、MMIO 領域に対するアクセスには命令のデコードが必要であるという点を悪用している。本論文では、通常、MMIO 領域へのアクセスは mov 命令などのメモリ参照命令に限られるという点に着目する。メモリ参照命令以外の不正な命令をエミュレートすることを避けるため、MMIO 領域に

¹ 慶應義塾大学
Keio University

アクセスする命令を限定する命令フィルタを導入することを提案する。この方式では、仮想マシンによる MMIO 領域へのアクセスの際にエミュレートされる命令をハイパーバイザが検証することで、MMIO 領域へのアクセスを行うことのない命令をフィルタアウトする。

提案方式を KVM (Linux Kernel 4.8) に実装した。Windows XP, 7, 10 及び Ubuntu 16.04 において MMIO 領域にアクセスする命令を調査し、MMIO 領域にアクセスすることのない命令の選定を行った。そのような命令によるアクセスを検知すると、その命令を未定義命令とみなし、エミュレーションを拒否する。なお、命令フィルタを導入するにあたってハイパーバイザを大きく改変する必要はなく、追加されたコード量は 159 LOC である。

この命令フィルタを導入した KVM 上で `sysenter` 命令をエミュレートさせる攻撃コードを実行したところ、`sysenter` 命令のエミュレーションを拒否することを確認した。また、命令フィルタの導入によるオーバーヘッドを測定したところ、最大 2.5% 程度であった。

本論文の構成を以下に示す。第 2 章では、ハイパーバイザによる MMIO 仮想化の問題点を述べる。第 3 章では、本研究の脅威モデルを述べる。第 4 章では、本研究の提案手法であるハイパーバイザへの命令フィルタの導入について説明する。第 5 章では、提案手法の実装の詳細を説明する。第 6 章では、提案手法の有用性をセキュリティ及びオーバーヘッドの観点からそれぞれ検証した実験について述べる。第 7 章では、本研究の関連研究について紹介する。第 8 章では、まとめ及び今後の課題について述べる。

2. ハイパーバイザによる MMIO 仮想化の問題点

エミュレートされる命令のすり替えによる脆弱性では、ハイパーバイザが MMIO 領域を仮想化する際に、命令のデコードを必要とすることを悪用している。MMIO 領域にアクセスした命令をすりかえることで、攻撃者は任意の命令エミュレーションを実行することができる。エミュレーションに関するコードに多くの脆弱性が存在するため、これは、セキュリティ上の脅威となる。ここではハイパーバイザの脆弱性とエミュレートされる命令のすり替えによる脆弱性が発生する原因となる MMIO 仮想化の方法について述べる。

2.1 ハイパーバイザのエミュレーションにおける脆弱性

ハイパーバイザではその上で複数の仮想マシンが動作するため、ハイパーバイザの脆弱性は一般的なソフトウェアの脆弱性と比べて影響力が強い。例えば、ハイパーバイザが DoS 攻撃を受けると、その上で動作する全ての仮想マシンも同様に DoS 攻撃を受けた状態になる。また、ハイパーバイザ上で攻撃コードが実行されると、他の仮想マシ

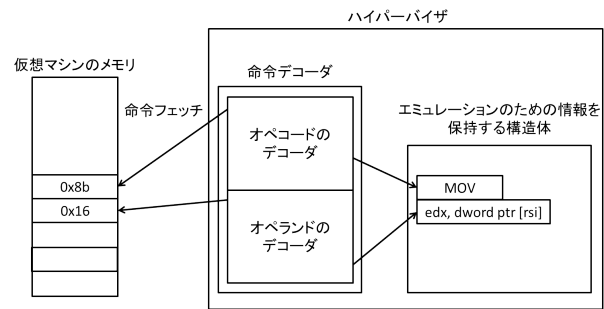


図 1 ハイパーバイザの命令フェッチ及びデコードの流れ

ンのセンシティブなデータを盗み見る VM Escape という攻撃も可能となる。

ハイパーバイザにおけるエミュレーションの脆弱性には次のものが知られている。CVE-2015-3456 は VENOM [9] とも呼ばれている。VENOM ではフロッピーディスクコントローラの仮想化を行うモジュールにバッファオーバーフローの脆弱性があったため、そのモジュールと同等の権限で任意のコードを実行することができる。CVE-2015-2151 [10] は Xen のセグメントを上書きする命令のエミュレーションに関する脆弱性である。この脆弱性は本来、セグメントを上書きする際に、オペランドがレジスタである場合、その上書きを無視するが、エミュレーションではそれを考慮していなかったことが原因である。その結果として、攻撃者の仮想マシンが他の仮想マシンの情報を覗いたりハイパーバイザをクラッシュさせることができる。さらに、KVM に関するエミュレーションの脆弱性として、CVE-2017-2583 [11], CVE-2016-9777 [12], CVE-2016-9756 [13], CVE-2014-8481 [14] や CVE-2014-3647 [15] が挙げられる。

2.2 ハイパーバイザによる MMIO 仮想化の方法

ハイパーバイザは仮想マシンの MMIO 領域へのアクセスをエミュレートする必要がある。これは、ハイパーバイザ上で複数の仮想マシンがデバイスを共有するため、デバイスを仮想化する必要があるためである。cpuid 命令や invlpg 命令などの命令をエミュレートする際には CPU の仮想化支援機構によってエミュレートする命令を取得することができる。しかし、MMIO 仮想化の場合、エミュレートする命令を仮想化支援機構によって取得することはできない。これは、MMIO 領域へのアクセスは特定の命令によって行われるものではないことが原因である。MMIO では各デバイスレジスタがメモリ空間の一部にマップされているため、OS が MMIO を用いるデバイスへのアクセスを行う際には、通常の方法でアクセスすることができる。そのため、仮想マシンのインストラクションポインタから MMIO 領域へアクセスした命令をフェッチ、デコードする必要がある。図 1 にその流れを示す。

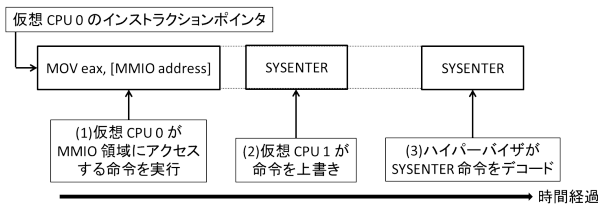


図2 命令のすり替えによって sysenter 命令をデコードする流れ

2.3 ハイパーバイザの MMIO 仮想化による任意命令エミュレーションの問題

ハイパーバイザが MMIO のエミュレートをするにあたって、図2のように命令のデコードが完了する前に他の仮想 CPU から MMIO 領域へアクセスした仮想 CPU のインストラクションポインタが指し示す内容を書き換えることで、ハイパーバイザに任意の命令をデコードさせることができる。エミュレーションされる頻度が高くなく、かつ、動作が複雑な命令はエミュレーションに不具合があることが多く、ハイパーバイザの脆弱性となりやすい。実際、CVE-2015-0239 [16] では sysenter 命令のエミュレーションに脆弱性が存在する。これは sysenter 命令が CPU が 16 bit モードであることを考慮していなかったことによる脆弱性で、仮想マシンへの DoS 攻撃や権限昇格を引き起こす可能性がある。なお、sysenter 命令がハイパーバイザによってエミュレートされる条件は限られており、物理 CPU が AMD であるにもかかわらず仮想 CPU が Intel に設定されている場合のみである。さらに、他の命令エミュレーションの不備による脆弱性として、CVE-2017-2596 [17], CVE-2017-2584 [18] や、CVE-2014-8480 [19] が挙げられる。

3. 脅威モデル

脅威モデルとして、攻撃者が仮想マシンを自由に操作することができる想定している。そのため、攻撃者は仮想マシン上で自由にアプリケーションを実行することや、MMIO 領域へのアクセスを行うことができる。なお、攻撃者が仮想マシンからハイパーバイザの脆弱性を攻撃することで権限昇格や DoS 攻撃を行うことを想定している。

4. 提案

本論文では仮想マシンの MMIO 領域へのアクセスに対して命令フィルタを導入することにより、エミュレートされる命令のすり替えによる脆弱性の回避手法を提案する。本論文で提案する命令フィルタでは MMIO 領域にアクセスする命令の種類が限られていることに着目し、MMIO 仮想化の際にハイパーバイザがエミュレートする命令を制限する。

4.1 MMIO 領域にアクセスを行う命令

しかし、MMIO 領域にアクセスする命令はその領域の特性上、デバイスの状態の変更あるいは確認を行うような命令に限られる。そのため、ハイパーバイザが MMIO 仮想化の際に、call 命令や jmp 命令のような命令をエミュレートしようとしたときには、エミュレートされる命令のすり替えが行われている可能性が高い。そのため、命令フィルタによってエミュレートする命令の制限を厳しくすることで、より多くのエミュレーションの脆弱性によるハイパーバイザへの攻撃を防ぐことができる。

実際、KVM 上で Windows XP, 7, 10 及び Ubuntu 16.04 を起動し、Ubuntu 16.04 上で UnixBench [20], ApacheBench [21], sysbench [22] を実行し MMIO 領域にアクセスする命令のログを取得した。そのログから、MMIO 領域へアクセスする命令を値をコピーする命令、値を比較する命令とその二つに該当しない or 命令の 3 種類に分類した。値をコピーする命令に該当するのは mov, movsb, movsd, stosd 命令の 4 種類である。また、値を比較する命令は cmp, test 命令の 2 種類が該当する。それぞれの命令の例は次のようになった。

- 値をコピーする命令
 - mov dword ptr [rsi], 0
 - rep movsb byte ptr [rdi], byte ptr [rsi]
 - rep stosd dword ptr [rdi], eax
- 値を比較する命令
 - cmp dword ptr [rax], 0x54464269
 - test byte ptr [rcx + 8], 0x20
- or 命令
 - or dword ptr [rsi + 0x10], 0x100ff

したがって、本提案の命令フィルタでは MMIO 領域の仮想化の際に、エミュレートする命令をこれら 3 種類の命令に制限する。

4.2 命令フィルタによる命令デコード結果の活用

命令フィルタでは MMIO 領域へのアクセスを行わない命令をフィルタアウトする。そのため、ハイパーバイザがエミュレートする命令の情報が必要である。命令フィルタがハイパーバイザのエミュレートする命令の情報を取得する方法は 2 種類ある。

まず、一つ目の方法としてハイパーバイザがエミュレートするために行う命令デコードの結果を命令フィルタで用いる方法が挙げられる。この方法では、最終的なデコード結果を用いて命令の検証を行うため、検証する命令と実際にエミュレートされる命令で食い違いが発生しないことが利点として挙げられる。一方で、ハイパーバイザの命令デコード部にある脆弱性を回避することができないことが欠点として挙げられる。

次に二つ目の方法として、命令フィルタ内で命令のデ

コードを行う方法が挙げられる。この方法では、ハイパーバイザのデコードに依存せずに命令のフィルタリングをするので、ハイパーバイザのデコード部にある脆弱性を回避できることが利点として挙げられる。しかし、命令フィルタでの命令検証後、ハイパーバイザの命令デコード前にエミュレートする命令をすり替えられた場合にはデコード部及びエミュレート部にある脆弱性を回避することができない。

したがって、命令フィルタではハイパーバイザの命令デコードの結果を用いて命令の検証を行う。なお、命令フィルタが命令の検証に用いる命令デコードの結果をオペコードのデコード結果に限定することができる。これによって、オペランドのデコードによる脆弱性を回避することができる。

5. 実装

本論文で提案する命令フィルタを Extended Page Table (EPT) が有効な x86-64 用の KVM (Linux Kernel 4.8) に実装した。ここでは KVM での MMIO 仮想化の実装方法と命令フィルタによる命令の検証方法について説明する。

5.1 概要

KVM は MMIO を仮想化するために、仮想マシンが MMIO 領域へアクセスした際に仮想マシンから KVM へ処理を移す VMExit を発生させる。MMIO 領域へのアクセスは通常のメモリへのアクセスと同じ方法でアクセスできるため、KVM では MMIO 領域のページテーブルエントリを読み書き不可、あるいは書き込み実行可にしておくことで、EPT violation あるいは EPT misconfig による VMExit を発生させて MMIO 領域へのアクセスを捕捉する。

VMExit の発生後、KVM は VMExit に関する情報が格納された Virtual Machine Control Structure から EPT violation あるいは EPT misconfig が発生した原因となるアドレスを取得し、そのアドレスが MMIO 領域であることを確認する。その後、MMIO 領域にアクセスした命令をインストラクションポインタからフェッチしデコードする。そのデコード結果を元に命令に対応するエミュレーションのハンドラを実行する。ハンドラの実行後、KVM が再び仮想マシンに VMEntry を行い、仮想マシンへ処理を移す。

5.2 命令フィルタによる命令の検証

命令フィルタでは MMIO 仮想化の際に MMIO 領域へのアクセスを行わない命令をフィルタアウトする。そのため、EPT violation あるいは EPT misconfig によって VMExit が発生し、仮想マシンの MMIO 領域へのアクセスがその原因と特定できた場合に、命令フィルタはハイ

表 1 命令の種類とエミュレーションのハンドラ

命令の種類	エミュレーションのハンドラ
値をコピーする命令	em_mov
値を比較する命令	em_cmp または em_test
or 命令	em_or

パーバイザがエミュレートする命令を検証する。

命令フィルタはハイパーバイザがエミュレートする命令の情報が必要なため、ハイパーバイザの命令のデコード結果を活用する。KVM での命令のデコード処理は主に接頭辞、オペコード、オペランドに分けられる。そのうちのオペコードのデコードのために、KVM ではテーブルに各オペコードに対するエミュレーションのためのフラグとハンドラを保持している。例えば、オペコード 0x0f34 の sysenter 命令はハンドラとして em_sysenter が登録されている。そして、オペコードをデコードすることで得られたフラグとハンドラを用いて、オペランドのデコード及びエミュレーションを行う。なお、4.1 節でも述べたように MMIO 領域にアクセスする命令は 3 種類に分類され、それらのエミュレーションのハンドラは表 1 のように対応付けられる。そこで、命令フィルタを KVM の命令デコード中のオペコードのデコード後に実装し、オペコードのデコード結果であるフラグとハンドラを用いて命令がそれら 3 種類の命令に該当するかを判断する。そのため、命令フィルタはハイパーバイザの接頭辞及びオペコードのデコードの結果を用いて命令が MMIO 領域にアクセスするものか検証する。MMIO 領域へアクセスする命令のハンドラは表 1 から em_mov, em_cmp, em_test, em_or のいずれかであることが分かる。命令フィルタでは、これらのハンドラが設定されている命令を KVM がエミュレートする場合にのみ命令のデコード及びエミュレーションを続行する。また、命令フィルタでは、接頭辞及びオペコードのデコード結果を用いるので、オペランドのデコード部の脆弱性は防ぐことが可能である。

命令フィルタでは、エミュレーションを拒否する際に、その拒否した命令を未定義命令として扱う。未定義命令として扱うことで、KVM はすり替えられた命令のエミュレーションを行わずに仮想マシンへ処理を戻すことができる。

命令フィルタは、KVM で #UD 例外をエミュレートすることで、エミュレートしなかった命令を未定義命令として扱っている。これにより、KVM から仮想マシンへ処理が戻った際に、仮想マシンでは #UD 命令が発生した状態になる。

6. 評価

命令フィルタのセキュリティ面及び性能面で実験を行った。実験環境は表 2, 3 のようになっている。

表 2 ソフトウェア環境

ソフトウェア名	バージョン
ホスト OS	Ubuntu 16.04 x86_64
ホスト Linux Kernel	4.8.1
QEMU	2.6.2
ゲスト OS	Ubuntu 16.10 x86_64
ゲスト Linux Kernel	4.8.0
ゲスト Apache	2.4.18
ゲスト gcc	6.2.0

表 3 ハードウェア環境

CPU	Intel Xeon CPU X5650 2.67 GHz
RAM	4 GB

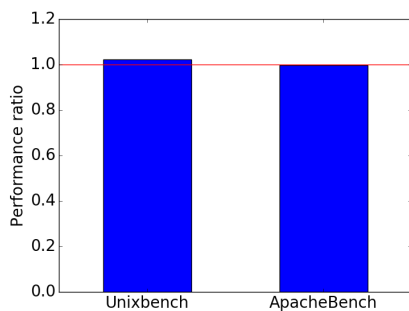


図 3 UnixBench 及び ApacheBench における命令フィルタの有無による性能比

6.1 セキュリティ評価

命令フィルタを実装した KVM 上で MMIO 領域へアクセスする命令を `sysenter` 命令にすり替えエミュレートさせる攻撃コードを実行したところ、未定義命令としてエミュレートし、`sysenter` 命令のエミュレートは行われなかった。したがって、命令フィルタによってエミュレートされる命令のすり替えによる脆弱性が回避されている。

4.2 節でも述べたように命令フィルタはハイパーバイザのデコード結果を用いるため、デコード部にある脆弱性を回避することはできない。しかし、命令フィルタが用いるデコード結果を一部に限定することで、その部分以外のデコード部の脆弱性を回避することができる。実際、デコード部にあった脆弱性のうち、CVE-2016-8630 [23] と CVE-2014-8481 [14] は命令フィルタによるフィルタリング後の部分にある脆弱性のため、エミュレートされる命令のすり替えによってこの脆弱性を引き起こそうとした場合、命令フィルタによって脆弱性が回避される。また、CVE-2014-8480 [14] もデコード部にあった脆弱性であるが、命令フィルタがホワイトリスト形式で命令を検証しているため、エミュレートされる命令のすり替えによってこの脆弱性を引き起こそうとした場合、命令フィルタによって脆弱性が回避される。

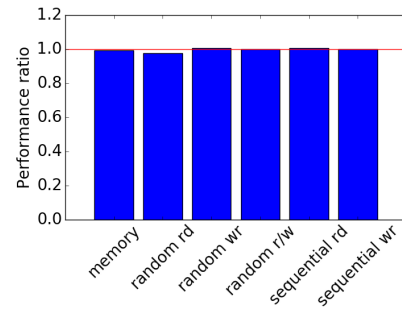


図 4 sysbench における命令フィルタの有無による性能比

6.2 オーバーヘッドの測定

命令フィルタの導入によるオーバーヘッドの測定のため、通常の KVM と命令フィルタを導入した KVM 上でベンチマークを実行した。評価に用いたベンチマークは UnixBench [20], Apache Bench [21], sysbench [22] である。その結果は図 3, 4 のようになった。命令フィルタの導入によるオーバーヘッドは最大で sysbench の random read の際の 2.5% となった。命令フィルタでフィルタリングを行う命令が MMIO 領域へアクセスした命令に限られていることとフィルタリングの処理自体が複雑でないことから、このようなオーバーヘッドになったと推測される。

7. 関連研究

ハイパーバイザの脆弱性について調査している研究として Deconstructing Xen [7] や Virtual CPU Validation [8] が挙げられる。Deconstructing Xen では Xen Security Advisories を調査し、Xen の脆弱性の原因となる部分、脆弱性の要因、脆弱性によって引き起こされる結果の観点から分析している。Virtual CPU Validation では物理 CPU のテストケースを仮想 CPU に適用することで、ハイパーバイザのテストを行い、その脆弱性を分析している。実際にテストケースを適用することで 6 個の脆弱性を発見している。そのうちの 하나가 `sysenter` 命令のエミュレーションの脆弱性である CVE-2015-0239 である。

ハイパーバイザの trusted computing base (TCB) を減らす研究として Deconstructing Xen, NoHype [24] や NOVA [4] が挙げられる。これらはモノリシックな設計となっているハイパーバイザを分割、あるいは削除することで TCB を削減し、ハイパーバイザの脆弱性を回避しようとしている。Deconstructing Xen では Xen の脆弱性がメモリ管理部分やコードエミュレーション部分に多いことに着目し、Xen を各仮想マシン毎に割り当て、Security monitor を追加することで、TCB の削減をしている。NoHype ではハイパーバイザを取り除き、物理資源をそれぞれの仮想マシンに専有するように割り当てることで、ハイパーバイザの脆弱性を回避している。NOVA はハイパーバイザをマイクロハイパーバイザとそれ以外に分けて、マイクロハ

ハイパーバイザのみを TCB とすることで TCB を削減している。

ハイパーバイザのランタイムなセキュリティの保護としては HyperSafe [25] や HyperVerify [26] が挙げられる。HyperSafe ではハイパーバイザの制御フローを制限するためにページテーブルへの書き込みを制限することと call 及び ret 命令でのジャンプ先を制御することを行う。HyperVerify では信頼できる仮想マシンを用意し、それがハイパーバイザの Non-control Data を監視することでランタイムにハイパーバイザを保護している。

8. まとめ

本論文では、ハイパーバイザ固有の脆弱性であるエミュレートされる命令のすり替えによる脆弱性を回避するためにハイパーバイザに命令フィルタを導入することを提案した。ハイパーバイザの脆弱性は一般的なソフトウェアの脆弱性と比べて致命的であるにもかかわらず、多くの脆弱性が報告されている。また、ハイパーバイザのエミュレート部分は脆弱性の原因となりやすい。これらのことから、エミュレートされる命令のすり替えによる脆弱性を回避することはハイパーバイザのセキュリティの向上につながる。

本論文では MMIO 仮想化の際にエミュレートされる命令を命令フィルタによってフィルタリングした。これはハイパーバイザでは MMIO 仮想化の際に、仮想マシンのインストラクションポインタから命令をフェッチし、デコードした後にエミュレーションを行うことから、エミュレートされる命令のすり替えによる脆弱性を引き起こすことが可能なためである。ud2 命令のように MMIO 仮想化と同様にエミュレーションを行うために仮想マシンのインストラクションポインタから命令をフェッチしデコードする必要がある命令も存在するので、それらに対しても命令フィルタによって命令の検証を行うことが今後の課題として挙げられる。

謝辞

本研究は JST, CREST の支援を受けたものである。

参考文献

- [1] : Google App Engine, <https://cloud.google.com/appengine/>.
- [2] : Amazon EC2, <https://aws.amazon.com/jp/ec2/>.
- [3] : KVM, http://www.linux-kvm.org/page/Main_Page.
- [4] Steinberg, U. and Kauer, B.: NOVA: A Microhypervisor-based Secure Virtualization Architecture, *Proceedings of the 5th European Conference on Computer Systems*, (EuroSys '10), New York, NY, USA, ACM, pp. 209–222 (online), DOI: 10.1145/1755913.1755935 (2010).
- [5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *SIGOPS Oper. Syst. Rev.*, Vol. 37, No. 5, pp. 164–177 (online), DOI:

- 10.1145/1165389.945462 (2003).
- [6] : Xen Security Advisories, <https://xenbits.xen.org/xsa/>.
- [7] Shi, L., Wu, Y., Xia, Y., Dautenhahn, N., Chen, H., Zang, B., Guan, H. and Li, J.: Deconstructing Xen, *The Network and Distributed System Security Symposium 2017*, (NDSS '17) (2017).
- [8] Amit, N., Tsafir, D., Schuster, A., Ayoub, A. and Shlomo, E.: Virtual CPU Validation, *Proceedings of the 25th Symposium on Operating Systems Principles*, (SOSP '15), New York, NY, USA, ACM, pp. 311–327 (online), DOI: 10.1145/2815400.2815420 (2015).
- [9] : VENOM, <http://venom.crowdstrike.com/>.
- [10] : CVE - CVE-2015-2151, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-2151>.
- [11] : CVE - CVE-2017-2583, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-2583>.
- [12] : CVE - CVE-2016-9777, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9777>.
- [13] : CVE - CVE-2016-9756, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-9756>.
- [14] : CVE - CVE-2014-8481, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-8481>.
- [15] : CVE - CVE-2014-3647, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3647>.
- [16] : CVE - CVE-2015-0239, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-0239>.
- [17] : CVE - CVE-2017-2596, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-2596>.
- [18] : CVE - CVE-2017-2584, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-2584>.
- [19] : CVE - CVE-2014-8480, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-8480>.
- [20] : Unix Bench, <https://github.com/kdlucas/byte-unixbench>.
- [21] : Apache Bench, <https://httpd.apache.org/docs/2.4/programs/ab.html>.
- [22] : sysbench, <https://github.com/akopytov/sysbench>.
- [23] : CVE - CVE-2016-8630, <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-8630>.
- [24] Keller, E., Szefer, J., Rexford, J. and Lee, R. B.: No-Hype: Virtualized Cloud Infrastructure Without the Virtualization, *Proceedings of the 37th Annual International Symposium on Computer Architecture*, (ISCA '10), New York, NY, USA, ACM, pp. 350–361 (online), DOI: 10.1145/1815961.1816010 (2010).
- [25] Wang, Z. and Jiang, X.: HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity, *IEEE Symposium on Security and Privacy*, IEEE, pp. 380–395 (online), DOI: 10.1109/SP.2010.30 (2010).
- [26] Ding, B., He, Y., Wu, Y. and Lin, Y.: HyperVerify: A VM-assisted Architecture for Monitoring Hypervisor Non-control Data, *Proceedings of the 2013 IEEE Seventh International Conference on Software Security and Reliability Companion*, SERE-C '13, Washington, DC, USA, IEEE Computer Society, pp. 26–34 (online), DOI: 10.1109/SERE-C.2013.20 (2013).