Vol.2022-OS-155 No.5
2022/5/26

ハイパーバイザにおける仮想デバイスの脆弱ウィンドウを短縮する不正 I/O フィルタ

瀬貫 真菜[1,a)]　石黒 健太[1,†1,b)]　河野 健二[1,c)]

概要：マルチテナント型のクラウド環境において，ハイパーバイザは仮想マシンの隔離と安全性を保証しているものの，ハイパーバイザにも多くの脆弱性が報告されており，特にデバイスエミュレータはその温床となっている．デバイスエミュレータはその複雑さから修正パッチの開発が容易ではなく，脆弱性の発見からパッチの適用までに長い時間を要することが多い．本論文では，デバイスエミュレータを対象に一時的に脆弱性を防ぐ手法を提案する．デバイスエミュレータの脆弱性を突く攻撃では，通常の I/O 処理では行われることのない I/O 要求を行うことが多い．そこで，脆弱性を突く不正攻撃に特有の I/O 要求を拒否する I/O フィルタの仕組みを導入する．I/O フィルタでは，特定の脆弱性を突くための特徴的な I/O 要求が記述でき，その記述に合致する I/O 要求を拒絶する．このフィルタはデバイスの内部構造を理解することなく記述可能であるため，修正パッチの作成より容易であることが期待できる．実際に CVE-2015-3456 (VENOM), CVE-2016-7909 など 4 個の脆弱性を防ぐフィルタの記述が可能であり，通常時の仮想マシンの動作を阻害することはなく，実行時オーバーヘッドも最大 8 %程度であることを示す．

1. はじめに

マルチテナント型のクラウド環境において，ハイパーバイザは仮想化や仮想マシンの管理を行なっている．同一の物理マシン上に立てられた仮想マシンは同一のハードウェアリソースを使用する．ハイパーバイザは仮想マシンの隔離と安全性を保証し，物理マシンの動作に影響が出ないように管理を行なっている．ハイパーバイザが仮想化している資源の 1 つに I/O がある．仮想マシンの Guest OS は物理マシンを直接操作することはできない．仮想マシンが I/O を要求すると仮想マシンの制御は Guest OS からハイパーバイザに移動する．その上で，ハイパーバイザの中のデバイスエミュレータがエミュレートしたり，実際のハードウェアの操作を Host OS に依頼したりしている．このような形で，仮想マシンが多重化されたデバイスを利用することによる競合が起こらないようになっている．

一方で，ハイパーバイザには脆弱性が存在している．その中でも，デバイスエミュレータはハイパーバイザにおける脆弱性の温床になっている．例えば，フロッピーディスクコントローラ (FDC) の脆弱性である VENOM (CVE-2015-3456) [1]，AMD PC-Net PCI II Ethernet Controller の脆弱性である CVE-2016-7909 [2]，ATI SVGA の脆弱性である CVE-2020-13800 [3] など特定のデバイスによらず報告されている．これによる問題点が 2 つある．1 つ目はハイパーバイザが攻撃されることによる危険性である．ハイパーバイザが攻撃を受けると，その物理マシン上で動作している仮想マシン全てが影響を受ける．VENOM の場合，この脆弱性が利用されると，ゲストクラッシュを引き起こし，任意のコードを実行できる可能性があると分析されている [1]．ハイパーバイザが保証している仮想マシンの独立性が，ハイパーバイザへの攻撃により侵害されてしまう．2 つ目は修正パッチがあてられるまで該当の脆弱性からハイパーバイザを守る手段がないことである．ハイパーバイザに脆弱性が発見された場合，通常はハイパーバイザに適切な修正パッチをあてることで対応する．しかし，ハイパーバイザの修正には時間を要する．まず修正パッチを開発するためには該当の脆弱性の分析が必要であり，さらに修正パッチを作成した後も実際にそれで問題が解決するか，正常に動作していた部分に影響を与えていないかを確かめるためのテストが必要だからである．Ngoc らは，KVM の 24 の脆弱性のうち，60 % が修正に 60 日以上要していたと述べている [4]．この期間は攻撃者が脆弱性を利用した攻撃を行うには十分な期間である．脆弱性が発見

[1] 慶應義塾大学
Keio University
[†1] 現在，法政大学
Presently with Hosei Uniersity
[a)] senuki@sslab.ics.keio.ac.jp
[b)] kenta.ishiguro.66@hosei.ac.jp
[c)] kono@sslab.ics.keio.ac.jp

されてから修正パッチがあてられるまでの空白期間の短縮が必要である．

本論文では，修正パッチがリリースされるまでの間，一時的にデバイスエミュレータの脆弱性からハイパーバイザを防御するフレームワークを提案する．まず，仮想マシンとハイパーバイザの間で新たに I/O をフィルタリングする層を導入する．そして，フィルタリングする層において，システム管理者が API を用いて不正な I/O の流れを指定してフィルタを作成し，同じ流れで行われた I/O 要求を拒否するというものである．デバイスエミュレータのコード自体を修正せずに脆弱性に対応することが可能であり，加えて，フィルタの作成及びフィルタの消去が簡単に行えるため，一時凌ぎに向いている．脆弱性を利用して攻撃するためには通常の I/O 要求には含まれない特定の I/O シーケンスが必要である．そのため，攻撃となる I/O を指定して防ぐようにすることで，通常の仮想マシンの動作を妨げることなく不正な I/O を拒否することが可能である．

フレームワークの実装にはプログラミング言語の Rust を使用した．Rust は型安全性とメモリ安全性を言語レベルで保証しているプログラミング言語である [5]．これにより，フィルタ層がメモリリークやオーバーフローといった問題を起こす可能性を限りなく無くすことができる．KVM + QEMU [6], [7] に対して，本論文のフレームワークでフィルタを作成したところ，VENOM，CVE-2015-5279，CVE-2016-7909，CVE-2020-13361 の 4 つの脆弱性に対してフィルタを記述することができた．そして，作成した VENOM のフィルタは VENOM の PoC コードを検知することができた．フィルタを導入したことによるオーバーヘッドは最大で 8 % であった．

本論文の構成について述べる．2 章ではハイパーバイザによるデバイスエミュレーションとその脆弱性について示す．3 章では本論文の脅威モデルを定義する．4 章では本論文が提案するフィルタリング機構について述べる．5 章ではフレームワークを使用して作成したフィルタの実験結果を報告する．6 章では関連研究について述べ，7 章で本論文のまとめを述べる．

2. デバイスエミュレーションと脆弱性

2.1 デバイスエミュレーション

仮想マシンによるデバイスの操作は，物理マシンと同様に，port I/O や memory-mapped I/O (MMIO) 領域へのアクセスにより行われる．どのデバイスがどの領域に対応するかは仮想マシンが稼働し始めたときに決定され，Guest OS はプロセスがアクセスした箇所から該当のデバイスについて I/O 要求の処理を行おうとする．しかし，仮想マシンの Guest OS は物理マシンの Host OS のように I/O を取り扱うことはできない．物理マシンに実在するハードウェア並びに I/O を 1 つの仮想マシンが独占すると，同じ物理マシン上で稼働している他の仮想マシンに影響を与えてしまうからである．

仮想ディスクを使用する際，インタフェースとして IDE を使用した場合は以下のような手順で I/O 要求の処理が行われる．

(1) 仮想マシンのプロセスが IDE の領域にアクセスして CPU が例外を発出する．
(2) 仮想マシンの制御が Guest OS からハイパーバイザに移る．
(3) ハイパーバイザのデバイスエミュレータが IDE の動作をエミュレートし，仮想ディスクを操作したことにより生じる物理ストレージデバイスの操作を Host OS に依頼する．
(4) 仮想マシンの制御が，ハイパーバイザから Guest OS に戻り，仮想マシンが再開する．

KVM + QEMU においては，KVM がハイパーバイザの役割を担い，デバイスエミュレーションを QEMU が行っている．QEMU は デバイスファイル /dev/kvm を通じて仮想マシンの情報を得ている．仮想マシンが I/O を要求すると，仮想マシンの稼働が一時中断し，仮想マシンの制御が Guest OS から KVM に移動する VM Exit が発生する．KVM は Virtual Machine Control Structure (VMCS) の情報から，仮想マシンが I/O を要求したことで VM Exit が発生したことを突き止める．その後，デバイスのエミュレートを QEMU に依頼するという流れで I/O 要求が処理されている．

2.2 デバイスエミュレータにおける脆弱性

2.1 節で述べたように，ハイパーバイザは仮想マシンが動作している物理マシンを直接制御できるため，ハイパーバイザにおける脆弱性は致命的である．しかし，ハイパーバイザにも脆弱性が存在している．その中でも，デバイスエミュレータの脆弱性が数多く報告されている．表 1 はデバイスエミュレータにおける脆弱性の一例を示したものである．特定のデバイスに偏らず，様々な種類のデバイスエミュレータで報告されていることがわかる．デバイスエミュレータの脆弱性が多い原因は，デバイスエミュレータの作成が難しいことにある．デバイスエミュレータを作成するためには，デバイスの動作や内部構造全体に精通している必要がある．一方で，デバイス内部のバッファや構造に関する情報はベンダーが公開していないことも多い．また，デバイスエミュレータでハードウェアの動作を再現しても，ハードウェアで起き得ないことがソフトウェア上では起きることがある．

デバイスエミュレータの脆弱性の例として，VENOM を取り上げて説明する．VENOM は QEMU の FDC，Intel 82078 FDC のエミュレータにおける脆弱性である．脆弱性の深刻さを表す CVSS v2.0 Score は 7.7 の High で高リ

表 1　デバイスエミュレータの脆弱性

CVE ID	CVSS v2.0 Score	デバイス
CVE-2015-3456	7.7	FDC
CVE-2015-5279	7.2	NE2000
CVE-2016-4439	4.6	SCSI
CVE-2016-7909	4.9	PCnet
CVE-2020-11102	6.8	Tulip
CVE-2020-13361	3.3	ES1370
CVE-2020-13800	4.9	ATI SVGA
CVE-2020-15863	4.4	XGMAC
CVE-2020-25085	4.4	SDCHI

```
#include <sys/io.h>

#define FIFO 0x3f5

int main() {
    int i;
    iopl(3);

    outb(0x0a,0x3f5); /* READ ID */
    for (i=0;i<10000000;i++)
        outb(0x42,0x3f5); /* push */
}
```

コード 1　VENOM の PoC コード

スクな脆弱性であるとされている．FDC 内の FIFO バッファにおけるバッファオーバーフローが原因で，Guest OS の管理者権限を持ったユーザが VENOM を悪用して攻撃を行うと，仮想マシンをクラッシュさせたり，物理マシン上で任意のコードを実行したりする可能性があるとされている [1]．コード 1 は VENOM の Proof-of-Concept (PoC) コード [8] を示している．この FDC は FIFO バッファを通して読み書きを行うことで操作する．コード 1 で使用されている READ_ID は FDC を操作するためのコマンドの 1 つである．FDC の仕様として，READ_ID コマンドを発行した後，コマンドの実行時間のエミュレートが行われる．この間，FIFO バッファはコマンドの実行が完了するまでアクセス禁止と定められており [9]，実際の FDC ではこの間に FIFO バッファへのアクセスは物理的にできないようになっている．一方で，QEMU においてはこの間でも物理的には FIFO バッファにアクセスすることができるため，大量に書き込むことでバッファオーバーフローを起こせるというものである．デバイスの内部構造を再現することの難しさ，そしてハードウェアとソフトウェアにおける動作の違いがこの脆弱性を生んだ原因である．

2.3　脆弱性対応における問題点

脆弱性が発見された場合，通常は修正パッチをあてるという対応がとられる．しかし，デバイスエミュレータのコードは複雑で修正が難しい．修正パッチの開発のためにまず脆弱性の分析が必要である．そして，修正パッチを開発した後も，その修正パッチで該当の脆弱性が解消されているか，新たな脆弱性を生み出していないか，修正パッチにより今まで正常に動いていた箇所に影響が出ていないかを検証する必要がある．デバイスエミュレータの修正にはある程度の時間が必要である．脆弱性に対応するために，ハイパーバイザの修正に要した時間として，Xen で平均 1，2 ヶ月，KVM で 71 日要していたという報告もなされている [4]．そのため，どうしても脆弱性が発見されてから修正パッチがあてられるまで空白期間が生じる．本論文では，この空白期間の短縮のために，デバイスエミュレータのコードを修正せずに，仮想マシンからの不正な I/O 要求を防ぐことを目的とする．

3.　脅威モデル

本論文においては，脅威モデルを以下のように定義する．まず，攻撃者は Guest OS の管理者権限を持っていると想定する．すなわち，攻撃者はゲスト環境において特権命令を自由に実行でき，仮想デバイスを自由に使うことができる．攻撃者が Guest OS の管理者権限を持っているという点について，攻撃者自身が元から Guest OS の管理者権限を持っていたかどうかは関係ないものとする．これは，攻撃者が仮想マシン内で動いているアプリケーションの脆弱性から Guest OS を乗っ取る可能性を考慮している．一方で，クラウドのプロバイダは悪意を持っていないものとする．

次に，仮想環境については完全仮想環境を想定する．完全仮想環境では仮想マシンは物理マシンと同じようにデバイスを利用するのに対して，準仮想環境ではあらかじめ改変して仮想化されたデバイスを実際のデバイスとは異なる形で利用する．また，準仮想環境ではデバイスドライバが，実際のハードウェアに対して使用されるデバイスドライバとは異なる場合がある．ただし，本論文で提案している手法は完全仮想環境，準仮想化環境であるかに関わらず適用することが可能である．準仮想化されたデバイスにおいても I/O 要求は行われるため，仮想マシンとハイパーバイザの間で I/O を監視して不正な要求を拒否するというアプローチは利用できるからである．

4.　I/O フィルタリング機構の設計と実装

4.1　概要

2 章で述べたように，脆弱性に対する修正パッチの開発には時間が必要である．その間，ハイパーバイザは該当の脆弱性に対して無防備になってしまう．そのため，脆弱性が発見されてから修正パッチがリリースされるまでの間，一時的にハイパーバイザを守る手段が必要である．修正パッチのリリース前に対応を行うためには，ハイパーバイ

ザのコードを変えずに適用可能であること，ハイパーバイザを守るための機構の適用・修正・撤去が簡単に行えることが重要である．

本論文では，ハイパーバイザの脆弱性として数多く報告されているデバイスエミュレータの脆弱性に着目し，修正パッチがリリースされるまでの間，一時的にデバイスエミュレータの脆弱性からハイパーバイザを防御するフレームワークを提案する．図 1 はフレームワークによって作成するフィルタリング機構の概要図を示している．仮想マシンとハイパーバイザの間で仮想マシンからの I/O をフィルタリングする層を導入する．このフィルタリング層では，仮想マシンが発行した I/O 命令の監視を行う．そして，正しい I/O はそのまま通す一方，不正な I/O はフィルタリング層において拒否し，デバイスエミュレータに実行させないようにするというものである．

システム管理者やデバイスエミュレータの開発者は，フィルタリング層において，フィルタとしてどのような I/O シーケンスが不正になるかを直接指定することができる．ここでは，攻撃要因となる部分を指定すれば良く，デバイスの仕様を完全に理解した上で厳密に指定する必要はない．VENOM において，コード 1 にある READ_ID コマンドを利用した攻撃を防ごうとした場合を想定する．この場合，FDC と READ_ID コマンドの一部の仕様についての知識は必要になる．例えば，READ_ID コマンドの仕様として実行にパラメータが 1 つ必要であることが挙げられる．また，READ_ID コマンド実行中に FIFO バッファはアクセス禁止になり，攻撃においては，FIFO バッファのステータスがアクセス禁止時に書き込まれているといった，脆弱性を利用した攻撃のどこが本来の FDC の動きと異なるかについても知る必要はある．一方で，READ_ID コマンドが FDC においてどのような働きをしているかや Intel 82078 FDC で他に 29 用意されているコマンドについての知識は必要としない．また，FDC 内部で FIFO バッファ以外にも多数定義されている状態についての知識は必要ない．脆弱性の要因となっている箇所の知識は必要になるものの，デバイス全体の仕様を詳細に知る必要はない．そのため，デバイスの内部構造や状態を詳細に知らなくとも，攻撃を防ぐためのフィルタを作成することが可能である．直接不正な I/O を指定して拒否しても通常のデバイス動作には影響を与えない．なぜなら，攻撃コードでは通常の I/O にはない流れでの I/O 要求が行われるからである．そもそも不正な I/O シーケンスと同じ I/O シーケンスが通常の I/O 要求でも使われるのであれば，エミュレータのテスト段階で脆弱性となる前に判明しているはずである．そのため，不正 I/O を直接禁止しても通常のデバイス操作が妨げずに仮想マシンを稼働させることができる．

フィルタリング層で行われていることはあくまでも仮想マシンからの I/O 要求の監視である．そのため，デバイス

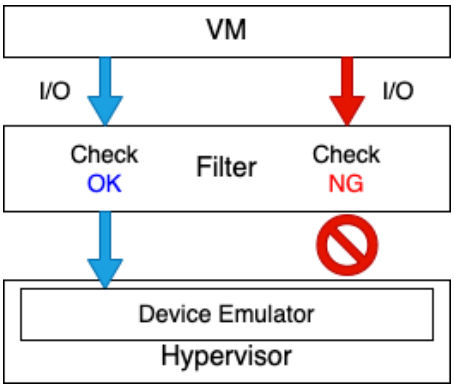

図 1 フィルタリング機構の概要図

エミュレータ並びにハイパーバイザのコードを改変する必要がない．また，仮想マシンの動作にも影響を与えることなく対応することが可能である．加えて，フィルタの作成及びフィルタの消去が簡単に行えるため，一時凌ぎに向いている．これまで，脆弱性が発見された場合は修正パッチを待つという対応しかとることができなかったが，本論文のフレームワークにより，デバイスエミュレータの脆弱性が発覚した時点で一時的な対処を行うことが可能になる．

4.2 I/O フィルタ

I/O フィルタは 4.1 節で述べたように，フィルタリング層において実際に I/O が不正か否か判断を行う部分である．フィルタはデバイスエミュレータの開発者並びにシステムの管理者が直接不正な I/O シーケンスを脆弱性ごとに直接指定して作成する．不正な I/O シーケンスを指定してフィルタを作成すると，同様の流れで I/O 要求が行われた場合にその事実がフィルタ内に記憶される．フィルタを作成時に指定された I/O 要求全てが行われると攻撃されていると判断し，該当の I/O 要求を拒否する．

コード 1 にある VENOM の READ_ID コマンドを利用した攻撃を防ぐフィルタを作成する場合について説明する．2 章でも述べたように，この攻撃は READ_ID コマンドが発行された後，FIFO バッファに大量の書き込みを行うことでバッファオーバーフローを起こすというものである．READ_ID コマンドは実行に，コマンド自体とパラメータ 1 つの FIFO バッファへの書き込みを必要とする．その後，FIFO バッファへのアクセスは禁止される [9]．FIFO バッファへのアクセスが許可されているか否かを示すステータスはエミュレータ内部で定義されている．このステータスは仮想マシンからは I/O で読み込みを行うことで判明する．通常 FIFO バッファに対するアクセスは，このステータスの状態を読み込み，アクセスが許可されていることを確認してから行われる．一方でコード 1 の攻撃コードでは，ステータスの確認なしに書き込みを行うことでバッファオーバーフローを引き起こしている．そのため，READ_ID コマンドが発行された後，FIFO バッファ

```
let venom_readid = Filter::new()
    .out(predicate::eq(0xa), 0x3f5)
    .out(predicate::always(), 0x3f5)
    .wait_until(
        Filter::new().inb_update(predicate
            ::function(|&x| x & 0x80 != 0),
             0x3f4),
        Filter::new().out(predicate::
            always(), 0x3f5),
    );
```

コード 2 VENOM に対応するフィルタ 1

表 2 API 一覧

系列	API	備考
new	new()	フィルタ作成
out	outb(value, port)	書き込み動作
	outw(value, port)	
	outl(value, port)	
	out(value, port)	
in	inb(port)	読み込み動作
	inw(port)	
	inl(port)	
	in_all(port)	
	inb_update(value, port)	結果付き
	inw_update(value, port)	読み込み動作
	inl_update(value, port)	
	in_update(value, port)	
wait	wait_until(cond_filter, ng_filter)	条件付き操作指定

のアクセスが許可されていることを仮想マシンが把握する前に FIFO バッファに書き込まれることを防ぐ必要がある．よって，VENOM の READ_ID コマンドを利用した攻撃を防ぐフィルタを作成する場合 READ_ID コマンド，必要なパラメータが書き込まれた後，仮想マシンが FIFO バッファのアクセスが許可されているか把握する前に書き込まれると不正だと指定する．実際に作成したフィルタをコード 2 に示す．

4.3 実装

表 2 は本論文で提案しているフレームワークにおける API 一覧を示している．基本的に I/O 要求に直接対応しているが，wait_until 系統のみ特殊で特定の条件を満たすまで特定の操作を禁止するという動作になる．コード 2 のように関数を呼び出して繋ぐと，引数でとった操作とそれまでのフィルタへの参照を記録する．フィルタを評価すると，要求された I/O が指定した不正な I/O と同じ流れを踏んでいるか確認する．

x86_64 で動作する KVM + QEMU [6], [7] の port I/O に対応するように実装を行った．フレームワークはプログラミング言語の Rust で作成した．Rust は型安全性とメモリ安全性を言語レベルで保証している [5]．これにより，フィルタリング機構がメモリリークやオーバーフローといった問題を起こす可能性を限りなく無くすことができる．また，Rust は動作速度が速いという特徴を持っている．ハイパーバイザのセキュリティを守りながら，フィルタリング層導入によるオーバーヘッドを最小限にすることができる．フレームワークはライブラリとして QEMU にダイナミックリンクして動かす．図 2 は仮想マシンから I/O を要求してからの処理の流れを示している．フィルタリングを含め，以下のように動作する．

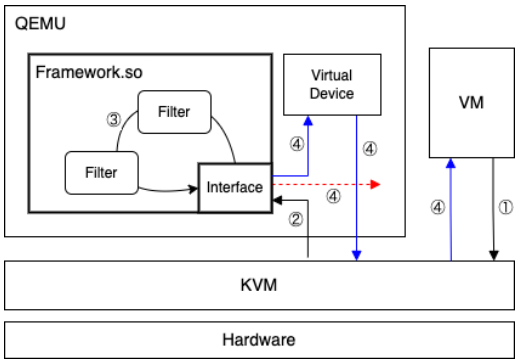

図 2 I/O 要求の処理の流れ

表 3 実験環境

Host OS	Ubuntu 18.04.5 LTS
Linux Kernel	4.15.0-153-generic
Host CPU	Intel Xeon Silver 4210 CPU @ 2.20GHz
Host Core	10
Host Memory	32 GB
QEMU Version	2.9.50
Guest OS	Ubuntu 18.04.6 LTS
Linux Kernel	4.15.0-166-generic
vCPUs	4
Guest Memory	8 GB

(1) 仮想マシンがデバイスを操作するために I/O を要求する．
(2) インタフェースから，仮想マシンが要求した動作 (読み込み，書き込みのどちらか)，アクセスしたアドレス，読み書きした値，操作しようとしたサイズの情報を得る．
(3) 登録されている Filter 全てについて，行われた I/O が，不正と指定した動作をしているか確認する．
(4) 正常な I/O であればデバイスエミュレータが通常通り実行し，不正な I/O であれば廃棄する．

5. 実験

5.2 節，5.3 節で使用した実験環境を表 3 に示す．

5.1 セキュリティ分析

VENOM 2 章でも述べたように，QEMU における Intel 82078 FDC のエミュレータにおける脆弱

性で，FDC 内の FIFO バッファにおけるバッファオーバーフローが原因である [1]．攻撃者が，FDC を操作するためのコマンドである READ_ID，もしくは DRIVE_SPECIFICATION_COMMAND を悪用すると，仮想マシンをクラッシュさせたり，物理マシン上で任意のコードを実行したりする可能性がある．READ_ID を利用した攻撃については 2 章で触れたため，ここでは DRIVE_SPECIFICATION_COMMAND を利用した攻撃について示す．DRIVE_SPECIFICATION_COMMAND はパラメータ 5 つを書き込むことで実行される．これらのパラメータのうち，5 つ目のパラメータを細工すると，FIFO バッファのリセット処理を回避する [10]．この後にコード 1 のように，FIFO バッファに大量に書き込むと，バッファオーバーフローを引き起こす．この脆弱性は，port I/O から FDC の FIFO バッファに書き込むことで攻撃を行う．そのため，本論文のフレームワークでコード 3 のように不正な I/O を直接指定して防ぐことが可能である．

```
let venom_drivespec = Filter::new()
  .out(predicate::eq(0x8e), 0x3f5)
  .out(predicate::always(), 0x3f5)
  .out(predicate::always(), 0x3f5)
  .out(predicate::always(), 0x3f5)
  .out(predicate::always(), 0x3f5)
  .out(predicate::function(|&x| x & 0x80
      == 0), 0x3f5);
```

コード 3 VENOM に対応するフィルタ 2

CVE-2015-5279 QEMU の NE2000 ネットワークカードのエミュレータにおける脆弱性である．悪用されると，仮想マシンによってパケット受信時に DoS 攻撃される，もしくは任意のコードを実行される可能性がある [11]．攻撃はデバイスレジスタ PSTART, PSTOP, BNRY, CURR に不適切な値を設定することで行われる．これら 4 つのデバイスレジスタの値は受信用バッファのインデックスを計算するために使用されている．これら 4 つのデバイスレジスタに不適切な値をセットすることで，パケット受信時に `ne2000_receive()` 内でバッファ用に確保されたメモリの範囲外へのアクセスを引き起こす．I/O でどのデバイスレジスタにアクセスするかを指定してから書き込む操作を行うことで，任意の変数に値をセットする．値のセットは port I/O から行うことができる．書き込まれる値が通常の I/O と異なるため，コード 4 のように，不正な I/O を指定して防ぐことが可能である．

CVE-2016-7909 QEMU における AMD PCnet-PCI II Single-Chip Full-Duplex Ethernet Controller である Am79C970A のエミュレータにおける脆弱性である．攻撃者はこの脆弱性を悪用することで，パケットの受信を通して DoS 攻撃を行う可能性がある [2]．攻撃はデバイスレジスタの CSR76 もしくは CSR78 に 0 をセットすることで行われる．これらのレジスタに 0 が設定されていると，パケット受信時に `pcnet_receive()` から無限ループを引き起こす可能性がある．どちらのデバイスレジスタも I/O でどのレジスタに書き込むかを設定した後，再度の I/O で指定したデバイスレジスタに任意の値を書き込むことで値をセットする．書き込みは port I/O から行うことができる．通常の操作においては CSR76，CSR78 に 0 が書き込まれることはない．そのため，これらのデバイスレジスタに 0 をセットする I/O を直接指定することで攻撃を防ぐことが可能である．作成したフィルタをコード 5 に示す．

```
let cve_2015_5279_pstart = Filter::new()
  .out(predicate::lt(0x40), 0xc000)
  .wait_until(
    Filter::new().out(predicate::ge(0x40),
        0xc000),
    Filter::new().out(predicate::function
        (|&x| x << 8 > 49152 as u32), 0
        xc001),
  );
```

コード 4 CVE-2015-5279 に対応するフィルタ

```
let cve_2016_7909_word = Filter::new()
  .outw(
    predicate::eq(0x4c)
      .or(predicate::eq(0xcc))
      .or(predicate::eq(0x4e))
      .or(predicate::eq(0xce)),
    0xc212,
  )
  .wait_until(
    Filter::new().outw(
      predicate::eq(0x4c)
        .or(predicate::eq(0xcc))
        .or(predicate::eq(0x4e))
        .or(predicate::eq(0xce))
        .not(),
      0xc212,
    ),
    Filter::new().out(predicate::eq(0), 0
        xc210),
  );
```

コード 5 CVE-2016-7909 に対応するフィルタ

CVE-2020-13361 QEMU における ENSONIQ AudioPCI ES1370 のエミュレータにおける脆弱性である．悪用されると，メモリの範囲外アクセスを引き起こす [12]．攻撃は，デバイスレジスタの FRAMECNT に不適切な値を設定することで行われる．FRAMECNT レジスタは

32 bit の値を格納するレジスタで，上位 16 bit と下位 16 bit に分けられて使用される．FRAMECNT に格納されている下位 16 bit の値が上位 16 bit の値より小さい場合，`es1370_transfer_audio()` 内でメモリアクセスの位置として不適切な値が算出される．FRAMECNT レジスタへの値のセットは 2 回の port I/O から行うことができる．攻撃時は通常の I/O と異なり，FRAMECNT レジスタにセットされる値が下位 16 bit の値が上位 16 bit の値より小さい．そのため，コード 6 のように FRAMECNT に不適切な値をセットする I/O を直接指定することで攻撃を防ぐことができる．

```
let cve_2020_13361_dac1 = Filter::new()
  .out(predicate::eq(0xc), 0xc10c)
  .wait_until(
    Filter::new().out(predicate::ne(0xc),
        0xc10c),
    Filter::new().outl(predicate::function
        (|&x| (x >> 16 & 0xffff as u32) > x
         & 0xffff), 0xc134),
  );
```

コード 6 CVE-2020-13361 に対応するフィルタ

CVE-2020-13800 QEMU における ATI SVGA のエミュレータにおける脆弱性である．デバイスレジスタの MM_INDEX に不適切な値が設定されると，関数実行時に無限再帰を引き起こす可能性がある [3]．MM_INDEX はレジスタやメモリアクセスにおけるインデックスの役割を果たしている．`ati_mm_read()`，`ati_mm_write()` は MM_INDEX の値によって再帰で呼び出される可能性がある．MM_INDEX が 7 以下のとき，永遠に再起を抜ける条件を満たさない可能性がある．MM_INDEX の値の設定を port I/O から行う方法は確認できなかった．そのため，現在の実装ではこの脆弱性を利用した攻撃を防ぐフィルタは作成できない．一方で，MM_INDEX の値の指定にはデバイスへのアクセスが必要であり，I/O が要求される．MMIO に対応することで QEMU から読み書きを行おうとしたアドレスと値の取得が可能である．そのため，4 章で示したアプローチを利用して防ぐことは可能である．

CVE-2020-15863 QEMU の XGMAC Ethernet Controller における脆弱性である．攻撃者がこの脆弱性を利用して攻撃を行うと，バッファオーバーフローを引き起こすことができ，QEMU のプロセスを破壊して DoS 攻撃を行ったり，ホストのシステム上で特権命令を実行したりする可能性がある [13]．パケット送信時，メモリからバッファに内容をコピーして送信を行なっている．このとき，メモリのデータを細工し，パケットの終端であることを示す値を含まないようにしておく．バッファを指していたはずのポインタが，バッファの範囲外を指すようになる．その結果，バッファとして確保されているメモリの範囲外への書き込みが行われる．この脆弱性はデバイスエミュレータの脆弱性ではあるものの，攻撃に I/O 要求を必要としない．そのため，本論文で提案しているフレームワーク及びアプローチで攻撃を防ぐことはできない．

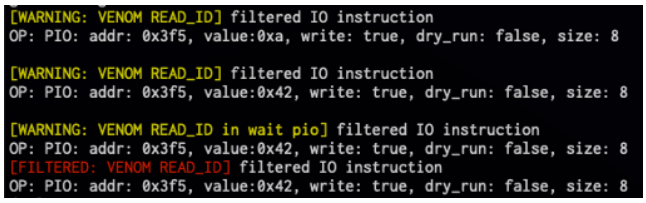

図 3 VENOM を悪用した攻撃がフィルタリングされた様子

5.2 セキュリティ評価

記述した VENOM のフィルタについて，不正な I/O を検知するかを確かめるために実験を行った．まず，VENOM の攻撃コードを検知するか確認するために以下の手順を実行した．

(1) 仮想マシンにフロッピーディスクをマウントした．
(2) VENOM の PoC コードのコード 1 を実行した．

次に，FDC における通常動作で何も検知されないかを確認するために以下の手順を実行した．

(1) 仮想マシンにフロッピーディスクをマウントした．
(2) ファイルをフロッピーディスク上で新規作成し，512 KB 書き込んだ．
(3) 作成したファイルに 512 KB 新たに内容を追加した．
(4) 作成したファイルを削除した．
(5) 仮想マシンからフロッピーディスクをアンマウントした．

その結果，VENOM の PoC コードを実行したときのみ図 3 のように攻撃コードが検知された．図 3 では，フィルタに登録された I/O が要求されたときに WARNING，フィルタに登録された一連の I/O と同じ流れで I/O 要求があり，攻撃されている可能性が高い場合に FILTERED と表示されるようになっている．一方で，FDC の通常操作では FILTERED と表示されることはなかった．フィルタによってデバイスの通常動作を妨げることなく，悪意ある仮想マシンからハイパーバイザを守ることが可能である．

5.3 オーバーヘッド測定

フィルタリング機構の導入による仮想マシンの動作への影響を調べるために，オーバーヘッドを計測した．Filebench [14] のワークロードのうち，fileserver，varmail，webserver の 3 つのワークロードを仮想マシンの IDE インタフェースからデフォルトの設定で動かした．測定は本論文のフレームワークなし (QEMU)，フレームワークを動作させてフィルタは 0 種類，フィルタを 11 種類追加したものそ

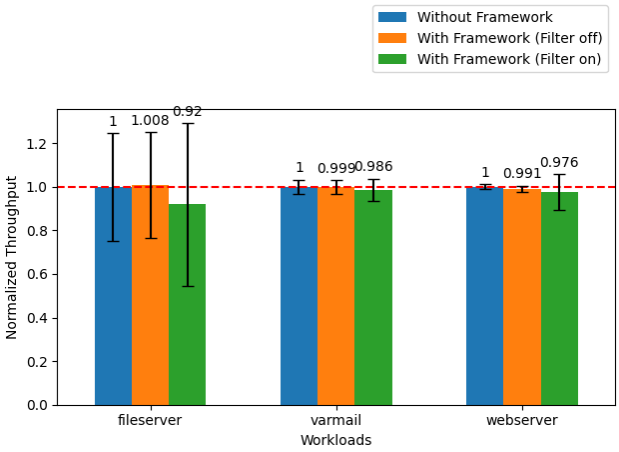

図 4 スループットの比較

れぞれで行い，スループットで比較を行った．Filebench は I/O のパフォーマンスを測定するワークロードである [14]．本論文では，仮想マシンからの I/O をフィルタリングするためのフレームワークを提案している．そのため，フレームワークを導入したことによるオーバーヘッドの測定には Filebench が適していると判断した．計測した結果は図 4 のようになった．フレームワークなしでのスループットを基準の 1 とし，それぞれの場合で比較を行っている．オーバーヘッドは fileserver のワークロードで最大の 8 % となった．仮想マシンから I/O 要求があると，制御が仮想マシンからハイパーバイザに移るため，CPU はレジスタの値を読み替える必要がある．この処理は CPU にとって重い処理であり，その上，2 回行う必要がある．対して本論文のフレームワークを導入したことで増える操作は Rust の関数呼び出しによるものである．関数呼び出しによるオーバーヘッドが，仮想マシンからの I/O 要求に伴う制御よりも処理として軽かったために，大きなオーバーヘッドとしては観測されなかったものと考えられる．

6. 関連研究

ハイパーバイザを仮想マシンから守る　悪意ある仮想マシンからハイパーバイザが攻撃されると，仮想マシンの隔離性が阻害される．Nioh [15] では本論文と同様に，デバイスエミュレータの脆弱性からハイパーバイザを守る手法を提案している．仮想マシンとハイパーバイザの間に I/O フィルタリング層を導入する．Nioh では仕様書に沿っていない I/O を不正とみなし，デバイスエミュレータが実行する前に廃棄するという手法をとっている．Nioh ではフィルタによって恒常的にハイパーバイザを守ることできるが，本論文のフレームワークでは一時的に守ることを主眼においている．一方で，Nioh ではフィルタ作成にあたって仕様書を読む必要があり，フィルタ作成に手間がかかるのに対し，本提案ではフィルタの作成は簡単に行うことができる．

ハイパーバイザの攻撃の要因を減らす　ハイパーバイザが複雑になるのに伴い，脆弱性が発生しやすくなっている．ハイパーバイザの機能を分割したり，削除したりすることによって脆弱性の元となるコードを減らそうとする取り組みがなされている．Nexen [16] では Xen の機能の一部を権限レベルを下げて仮想マシン毎のスライスに分割し，悪意ある仮想マシンが他の仮想マシンに影響を与えないようにしている．CloudVisor [17] は *Nested Virtualization* の考え方を取り入れ，ハイパーバイザをユーザレベルで実行するアプローチをとっている．ハイパーバイザの機能をリソースマネジメントと軽量なセキュリティモニタに分割し，ホストにおける特権をセキュリティモニタの Cloudvisor のみに持たせる．物理マシンの特権命令を使用するセキュリティモニタを通るようにすることで，ハイパーバイザ並びに仮想マシンの要求の正当性を保証している．CloudVisor-D [18] では CloudVisor で多発した VM Exit の回数を減らすため，Cloudvisor におけるセキュリティモニタをさらに Guradian-VM と nested hypervisor に分割している．Min-V [19] は仮想マシンで使用されるデバイスの種類が限られていることに着目し，ブート時に使用できるデバイスを減らすことで仮想マシンが攻撃される要因を少なくしている．本論文では脆弱性に対して簡単に対応するために，ハイパーバイザや仮想マシンのシステムには手を加えない手法を提案している．

攻撃可能な期間を短縮する　脆弱性対応にあたっては，該当の脆弱性のパッチ修正を待つのが一般的である．そのため，パッチがあてられるまでの期間は該当の脆弱性に対してハイパーバイザは無防備な状態となってしまう．HyperTP [4] は脆弱性対応の手法としてハイパーバイザの置き換えを提案している．Xen と KVM に共通している脆弱性が少ないことを利用して一時的にハイパーバイザの置き換えを行うことで空白期間の短縮を図っている．

7. まとめ

ハイパーバイザの脆弱性は致命的であるにも関わらず，数多くの脆弱性が存在している．その中でもデバイスエミュレータの脆弱性が数多く報告されている．また，脆弱性に対しては修正パッチを待つしか対応がなく，空白期間が生じていた．本論文では，デバイスエミュレータの脆弱性を悪用した攻撃からハイパーバイザを守るために不正 I/O を簡単に防ぐためのフレームワークを提案した．ハイパーバイザと仮想マシンの間でフィルタリングする層を導入することで，ハイパーバイザを修正することなく対応する．そして，不正な I/O が正常の I/O とは異なり，特定の手順が求められることに着目し，不正な I/O シーケンスを直接指定することで防止するというものである．本論文では KVM + QEMU の port I/O に対応するように実装

し，セキュリティとパフォーマンスの 2 つの側面で実験を行った．その結果，VENOM を悪用した攻撃を検知することが確認できた．また，オーバーヘッドは最大で 8 % であることがわかった．

謝辞 本研究は，JST，AIP 加速課題 JPMJCR22U3 の支援を受けたものである．

参考文献

[1] National Institute of Standards and Technology: CVE-2015-3456, National Institute of Standards and Technology (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2015-3456⟩ (accessed 2022-03-21).

[2] National Institute of Standards and Technology: CVE-2016-7909, National Institute of Standards and Technology (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2016-7909⟩ (accessed 2022-03-21).

[3] National Institute of Standards and Technology: CVE-2020-13800, National Institute of Standards and Technology (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2020-13800⟩ (accessed 2022-03-21).

[4] Ngoc, T. D., Teabe, B., Tchana, A., Muller, G. and Hagimont, D.: Mitigating vulnerability windows with hypervisor transplant, *EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021* (Barbalace, A., Bhatotia, P., Alvisi, L. and Cadar, C., eds.), ACM, pp. 162–177 (online), DOI: 10.1145/3447786.3456235 (2021).

[5] : Rust, Rust (online), available from ⟨https://www.rust-lang.org/⟩ (accessed 2022-03-21).

[6] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: KVM: the Linux Virtual Machine Monitor, *In Proceedings of the 2007 Ottawa Linux Symposium (OLS'-07* (2007).

[7] Bellard, F.: QEMU, a Fast and Portable Dynamic Translator, *Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference, April 10-15, 2005, Anaheim, CA, USA*, USENIX, pp. 41–46 (2005).

[8] : Openwall, Openwall (online), available from ⟨https://www.openwall.com/lists/oss-security/2015/05/13/29⟩ (accessed 2022-03-24).

[9] Intel: *82078 44 PIN CHMOS SINGLE-CHIP FLOPPY DISK CONTROLLER* (1995).

[10] CrowdStrike: VENOM Vulnerability Details, CrowdStrike (online), available from ⟨https://www.crowdstrike.com/blog/venom-vulnerability-details/⟩ (accessed 2022-04-05).

[11] National Institute of Standards and Technology: CVE-2015-5279, National Institute of Standards and Technology (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2015-5279⟩ (accessed 2022-03-21).

[12] National Institute of Standards and Technology: CVE-2020-13361, National Institute of Standards and Technology (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2020-13361⟩ (accessed 2022-03-21).

[13] National Institute of Standards and Technology: CVE-2020-15863, National Institute of Standards and Technology (online), available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2020-15863⟩ (accessed 2022-04-07).

[14] : filebench/filebench File system and storage benchmark that uses a custom language to generate a large variety of workloads., filebench (online), available from ⟨https://github.com/filebench/filebench⟩ (accessed 2022-04-07).

[15] Ogasawara, J. and Kono, K.: Nioh: Hardening The Hypervisor by Filtering Illegal I/O Requests to Virtual Devices, *Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017*, ACM, pp. 542–552 (online), DOI: 10.1145/3134600.3134648 (2017).

[16] Shi, L., Wu, Y., Xia, Y., Dautenhahn, N., Chen, H., Zang, B. and Li, J.: Deconstructing Xen, *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, The Internet Society, (online), available from ⟨https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/deconstructing-xen/⟩ (2017).

[17] Zhang, F., Chen, J., Chen, H. and Zang, B.: CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization, *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011* (Wobber, T. and Druschel, P., eds.), ACM, pp. 203–216 (online), DOI: 10.1145/2043556.2043576 (2011).

[18] Mi, Z., Li, D., Chen, H., Zang, B. and Guan, H.: (Mostly) Exitless VM Protection from Untrusted Hypervisor through Disaggregated Nested Virtualization, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020* (Capkun, S. and Roesner, F., eds.), USENIX Association, pp. 1695–1712 (2020).

[19] Nguyen, A., Raj, H., Rayanchu, S. K., Saroiu, S. and Wolman, A.: Delusional boot: securing hypervisors without massive re-engineering, *European Conference on Computer Systems, Proceedings of the Seventh EuroSys Conference 2012, EuroSys '12, Bern, Switzerland, April 10-13, 2012* (Felber, P., Bellosa, F. and Bos, H., eds.), ACM, pp. 141–154 (online), DOI: 10.1145/2168836.2168851 (2012).