キャッシュのモニタリングによるキャッシュサイドチャネル攻撃の検知

嶋田 有佑[1,a)]　河野 健二[1,b)]

概要：クラウド環境は仮想マシン間で資源を隔離し，ある仮想マシンが他の仮想マシンの情報にアクセスできない．しかしこれら仮想マシンは同じ物理ハードウェア上で動作し，物理資源を共有するため様々なサイドチャネル攻撃が可能である．代表的なものに CPU キャッシュを利用したキャッシュサイドチャネル攻撃があり GPG 暗号などの秘密鍵を復元できることが知られている．本論文ではキャッシュサイドチャネル攻撃によるキャッシュミスの様子が特徴的であることを用いて攻撃を検知する手法を提案する．従来の攻撃手法の多くはキャッシュ参照とキャッシュミスを多く生じるためこれらの総量を計測することで攻撃を検知できる．しかし攻撃手法の中でもステルス性の高い Flush+Reload はキャッシュ参照とキャッシュミスを多く生じないため攻撃検知が難しい．本論文では Flush+Reload のアルゴリズムに着目し，攻撃するプロセスのキャッシュミスの総量ではなく，キャッシュミスのパターンを特徴として用いることで攻撃を検知する．またサイドチャネル攻撃を受けているプロセスが通常より多くのキャッシュミスを発生することから攻撃を検知できることを示す．

キーワード：サイドチャネル攻撃，Flush+Reload，CPU キャッシュ，攻撃検知

Yusuke Shimada[1,a)]　Kenji Kono[1,b)]

1. はじめに

Amazon EC2 [1] や Microsoft Azure [2] に代表されるクラウド環境の普及に伴い，コンピュータ資源を効率的に使用できるようになっている．クラウド環境において，ハイパーバイザがコンピュータの物理資源を仮想化し，仮想化したコンピュータ資源を複数の仮想マシン（以降，VM と記述）に割り当てることでマルチテナント環境を実現している．例えば同一のプロセッサ上であったとしても，VM ごとに CPU コアを時分割で割り当てることで，VM 間で資源の分離によるアイソレーションを達成する．このアイソレーションにより，仮に複数のユーザが同一の物理ホスト上で VM を動かしたとしても，ある VM は他の VM の管理する情報にアクセスすることはできない．

しかし，これらの VM が同じ物理ハードウェア上で動作し，物理資源を共有する場合，共有される様々な物理資源がサイドチャネルとなり，ハイパーバイザによる強固なアイソレーションを回避してしまうサイドチャネル攻撃が可能となる．

代表的なものとして CPU のキャッシュをサイドチャネルとして利用するキャッシュサイドチャネル攻撃が存在する．キャッシュサイドチャネル攻撃の手法は様々であり，新たな攻撃手法も次々に研究されている（e.g., [3–5]）．代表的な攻撃手法としては Prime+Probe 攻撃や Flush+Reload 攻撃が存在し，過去の研究により標的のプロセスが GnuPG や ElGamal などの暗号ソフトウェアによる処理を行なった際に，攻撃者がこれらの秘密鍵を復元することに成功している [3, 6–11]．

Page-fault や branch prediction など，新たに様々なサイドチャネルを利用した攻撃手法 [12–14] が提唱される中で，依然として CPU キャッシュは強力なサイドチャネルとして利用され続けている．最近では暗号ソフトウェアの秘密鍵を標的として攻撃を行うだけではなく，標的のパスワードなどの機密情報を攻撃者に秘密裏に送る手段としても，こうしたキャッシュサイドチャネル攻撃の手法を利用した研究が存在する．実際に 2018 年の Spectre [15] と

[1] 慶應義塾大学
Keio University
[a)] shima_yu_1993@sslab.ics.keio.ac.jp
[b)] kono@sslab.ics.keio.ac.jp

Meltdown [16] においても Flush+Reload 攻撃をデータを送る手段として利用しており，こうした攻撃が社会に与える影響は極めて甚大なものとなっている．

このようなキャッシュサイドチャネル攻撃を検知，防御することは一般的に困難である．キャッシュサイドチャネル攻撃に対する検知，防御手法としてはいくつかのアプローチが存在し，CPU パフォーマンスカウンタを用いるソフトウェアベースの手法は，代表的な検知手法の一つである．

例えば，攻撃を行うプロセスのキャッシュ参照あるいはキャッシュミス回数の総量を測定し，スレッショルドを超えるような場合に攻撃であると判断する方法がある [4]．実際に代表的なキャッシュサイドチャネル攻撃手法である Prime+Probe 攻撃は，その攻撃手法の特徴からキャッシュ参照とキャッシュミスの回数が通常のプロセスに比べ極めて多くなることが知られている．

しかしながらこの検知手法ではステルス性が高いとされるキャッシュサイドチャネル攻撃を検知することは難しい．例えば Flush+Reload 攻撃では特定のいくつかのアドレスに対してのみメモリアクセスとキャッシュ階層からのフラッシュを行うため，攻撃を行う過程でキャッシュ参照とキャッシュミスの回数が通常のプロセスと比較して突出して多くなることはない．また，2016 年に新しく提唱された Flush+Flush 攻撃 [4] は攻撃過程においてメモリアクセスを一切行わないため，攻撃プロセスのキャッシュ参照をパラメータとした検知手法 [17,18] での検知は難しい．こうしたステルス性の高い攻撃手法を検知するためには別のアプローチが必要となる．

本論文では，2 種類のアプローチにより，これらのステルス性の高いキャッシュサイドチャネル攻撃を検知する手法を提案する．一つ目は，Flush+Reload 攻撃において，攻撃プロセスにより生じるキャッシュミスの仕方が特徴的であることに着目し，この性質を利用する手法である．先に述べた通り，Flush+Reload 攻撃に対しては，攻撃プロセスにより生じるキャッシュ参照とキャッシュミス回数の総量を測定しても他のプロセスと区別することは難しく，攻撃を検知できない．そこで，Flush+Reload 攻撃のアルゴリズムに着目し，一定の時間間隔ごとにキャッシュ参照およびキャッシュミス回数を測定することにより効果的に攻撃を検知することができる．

二つ目は，攻撃を受けているプロセスによるキャッシュミス回数が通常時よりも多くなるという特徴を利用した検知手法である．Flush+Reload 攻撃や Flush+Flush 攻撃を受けているプロセスは，他のプロセスが同時に動いている場合や，自身のプロセスのみが動いている場合と比較して，自身のプロセスによるキャッシュミス回数が多くなる．そこで，GnuPG などの暗号ソフトウェアによるセンシティブな処理を行うプロセスのキャッシュミス回数を測定することにより，攻撃を行っているプロセスが存在することを検知することができる．

本論文の構成を以下に示す．2 章では，近年の Intel CPU のキャッシュ階層構造および，CPU キャッシュをサイドチャネルとしたキャッシュサイドチャネル攻撃について述べる．3 章では，本研究の関連研究として，こうしたキャッシュサイドチャネル攻撃に対する既存の防御，検知手法について述べる．4 章では，本論文の一つ目の提案として，キャッシュサイドチャネル攻撃の中でもステルス性の高いとされる Flush+Reload 攻撃を検知する手法を提案する．5 章では，本論文の二つ目の提案として，4 章とは別の手法により Flush+Reload 攻撃およびさらにステルス性の高い Flush+Flush 攻撃を検知できることを提案する．6 章で本論文をまとめる．

2. 背景

2.1 CPU のキャッシュ階層およびキャッシュサイドチャネル攻撃

メモリアクセスに限っても，その過程で共有される様々な物理資源にサイドチャネルが存在することが知られている．Wang らの研究 [13] によれば，CPU 内の TLB やキャッシュ，ページテーブルエントリの他，DRAM に至るまでメモリアクセスを行う過程において様々なサイドチャネルが存在する．実際に DRAM をサイドチャネルとした攻撃も数多く提唱されている [19,20]．

本論文では CPU キャッシュをサイドチャネルとして利用したキャッシュサイドチャネル攻撃のみを対象としている．キャッシュサイドチャネル攻撃は原則として，データをメインメモリから読み出す場合と CPU キャッシュから読み出す場合で時間差が生じることを利用する．

メインメモリへアクセスする場合には大きなアクセス時間がかかるため，近年のプロセッサは内部に高速で小容量のメモリとしてキャッシュを用いている．近年のプロセッサのキャッシュは階層構造になっており，代表的な Intel 社のプロセッサは 3 段階になっているものが多い．最も上位，つまりプロセッサのコアに近いものが L1 キャッシュであり，最も容量が小さく，アクセス時間が短い．3 段階のキャッシュの場合，L1, L2 キャッシュは各コアごとに固有のものを保持しており，L3 キャッシュ，あるいは Last level キャッシュ，つまり LLC はすべてのコアで共有される共有キャッシュとなっている．

クラウド環境が普及してからは異なるユーザの VM 間で共有される LLC をサイドチャネルとした攻撃手法が主流となっている．LLC をサイドチャネルとすることで，仮にクラウドプロバイダが VM の配置ポリシーとして，異なるユーザの VM どうしを同じ CPU コアに割り当てることを避けていたとしても，異なるコア間での攻撃が可能である．

キャッシュサイドチャネル攻撃が対象とする機密情報としては，主に復号化や署名などの暗号化処理において使用されるユーザの秘密鍵であったが，他の種類のアプリケーションに対して攻撃を行う研究も存在する [21]．こうした機密情報は，秘密鍵の値などに依存した制御フローやデータフローにより，標的のユーザによるキャッシュ使用パターンとして攻撃者が観測することでリークしてしまう．攻撃者は共有キャッシュ内のデータを操作することにより，標的のキャッシュ使用パターンを推測し，得られたパターンから標的の機密情報を復元することができる．

2.2 代表的なキャッシュサイドチャネル攻撃

ここではキャッシュサイドチャネル攻撃の代表的な手法として，Prime+Probe 攻撃，Flush+Reload 攻撃そして Flush+Flush 攻撃について説明する．

2.2.1 Prime+Probe 攻撃

Prime+Probe 攻撃の手法は T.Zhang ら [18] によれば，2005 年に C.Percival の研究 [6] で初めて提唱され，その後クラウド環境でも実行できるように改良されてきた [9,11,22]．攻撃の流れの概略は次の通りである．

(1) Prime: 攻撃者は共有キャッシュセットを自身のデータやコードで埋める．
(2) Wait: 攻撃者は一定時間待機する．
(3) Probe: 再び攻撃者は Prime フェーズと同じデータおよびコードにアクセスし，アクセスにかかる時間を測定する．

攻撃者が待機している間に標的のプロセスが，Prime フェーズに使用したデータやコードと同じキャッシュセットにのるようなデータやコードにアクセスした場合，Prime フェーズで攻撃者がアクセスしたデータやコードの一部はキャッシュから追い出される．Probe フェーズでは，測定した時間があらかじめ定めておいたスレッショルドよりも大きい場合，キャッシュミスが発生していることがわかり，標的とするプロセスによるアクセスがあったことがわかる．

近年の攻撃ではクラウド環境を想定し，攻撃者のプロセスと標的のプロセスは異なる CPU コアで動いていると仮定している．そのため L1, L2 キャッシュは共有されておらず，共有キャッシュは LLC のみである．LLC はセットアソシエイティブ構造であり，容量自体も大きい．そのため，Prime フェーズでは攻撃者は最低でも標的とするデータやコードがのるキャッシュセット全体を埋めておく必要がある．キャッシュスライスなどを実装した近年の LLC の複雑さを考慮すると，確実に攻撃を行うためにはいくつものキャッシュセットを埋めておくことが必要 [11] であり，一回のフェーズでアクセスするデータやコードの数は多くなる．

2.2.2 Flush+Reload 攻撃

Flush+Reload 攻撃の手法自体は 2011 年の Gullasch らの研究 [8] により初めて提唱され，その後もクラウド環境など様々な仮想化環境で攻撃を検証する研究が行われてきた [3,4,10,21]．この攻撃は，異なる VM 間あるいはプロセス間で同一のメモリページを共有する必要がある．攻撃者はこの共有されたページの中から，標的プロセスによるキャッシュの使用パターンを解析する上で有効なデータあるいはコードのアドレスを選び出す．攻撃の流れは次の通りである．

(1) Flush: 攻撃者は選択したアドレスのメモリをすべてのキャッシュ階層からフラッシュする．
(2) Wait: 攻撃者は一定時間待機する．
(3) Reload: 攻撃者はフラッシュしたアドレスのメモリにアクセスし，アクセスにかかる時間を測定する．

Reload フェーズにおいて，測定した時間があらかじめ定めておいたスレッショルドより小さい場合，攻撃者が待機している間に，標的のプロセスにより対象のアドレスのメモリがアクセスされたことがわかる．

この攻撃手法のメリットは Prime+Probe 攻撃と比較して対象のアドレスに絞ってアクセスをすれば良いため，アクセスするデータの数が少なく，高速であり，ノイズが少ないという点である．攻撃を行うためには，攻撃者が標的のプロセスとアドレス空間を共有する必要があるが，近年のクラウド環境においては，コンピュータ資源をより効率的に利用するために memory deduplication などを行うことが知られており，こうした攻撃の条件は十分に充足することが可能である [10]．

2.2.3 Flush+Flush 攻撃

2016 年に提唱されたこの攻撃手法は，Flush+Reload 攻撃のそれと類似している．

(1) Flush: 攻撃者は選択したアドレスのメモリをすべてのキャッシュ階層からフラッシュする．
(2) Wait: 次に攻撃者は一定時間待機する．
(3) Flush: 攻撃者は再び同じアドレスに対してフラッシュを行い，フラッシュにかかる時間を測定する．

二度目の Flush フェーズにおいて，測定した時間があらかじめ定めておいたスレッショルドより大きい場合，攻撃者が待機している間に，標的のプロセスにより対象のアドレスのメモリがアクセスされたことがわかる． Intel の CLFLUSH 命令 により対象のデータをキャッシュ階層からフラッシュするのにかかる時間は，そのデータがキャッシュ階層に存在する場合，データがキャッシュ階層に存在しない場合よりも大きな時間がかかる [4]．この事実を利用することで標的のプロセスによるアクセスを検知することができる．

この手法はその攻撃過程において一切のメモリアクセスを行わないため，後述するような既存の検知手法では検知

が困難であるという点で，極めてステルス性が高い攻撃である．

3. 既存の防御・検知手法と問題点

キャッシュサイドチャネル攻撃に対する既存の防御および検知手法としては主に次のようなアプローチが存在する．

3.1 共有キャッシュの分割

共有の CPU キャッシュを VM ごとに分割し，他の VM のプロセスから干渉を受けないようにする [23,24]．分割の仕方としてキャッシュのセットあるいはウェイごとに分ける方法が存在する．例えば，それぞれの CPU コアに対して別々のキャッシュ領域を割り当てることを可能とする Intel CPU の拡張機能である Intel CAT [25] を用いたハードウェアベースの手法や，OS やハイパーバイザーを改良し，VM ごとにキャッシュ内で衝突が起こらないようにページを割り当てるなどのソフトウェアベースの手法が存在する．この手法はキャッシュサイドチャネル攻撃に対して包括的な防御手法であるものの，クラウドプロバイダは導入に大きなコストがかかるという問題点が存在する．

3.2 暗号ソフトウェアの改良

キャッシュサイドチャネル攻撃の典型として，暗号ソフトウェアの実行トレースから秘密鍵を復元するため，暗号ソフトウェアの実装を改良することで，攻撃者が正確なトレースを行うことを困難にし，鍵の復元を防ぐ手法が考えられてきた [6,7,26]．しかし，暗号ソフトウェアごとに対応する必要があり，また，新しい実装に適応した攻撃手法も提唱される [11] ため，メンテナンスを含め極めてコストがかかる一方で効果が限定的なアプローチである．

3.3 時間計測の困難化

従来のキャッシュサイドチャネル攻撃は，そのほとんどが時間計測により標的のプロセスによるメモリアクセスを確認する必要がある．そのため，CPU クロックに基づく時刻を得る RDTSC 命令などによる時間計測を制限することにより，こうした攻撃を防ぐ方法が研究されてきた [27,28]．しかし，近年では Prime+Abort 攻撃に代表されるように時間計測を必要としない攻撃手法が提唱されており，このアプローチでは防ぐことのできない攻撃が増えてくると想定される．

3.4 CPU パフォーマンスカウンタの利用

ソフトウェアレベルでのアプローチとして，CPU パフォーマンスカウンタの測定によって攻撃を検知する研究が存在する [4,17,18]．Prime+Probe 攻撃では大量のデータにアクセスすることからキャッシュの参照およびキャッシュミス回数の総量が通常のプロセスと比較して多くなることが知られている．この特徴を利用して Prime+Probe 攻撃を行うプロセスを検知することができる．しかしながら Flush+Reload 攻撃はその攻撃過程において，Prime+Probe 攻撃と比較してキャッシュ参照およびキャッシュミス回数の総量が多くなく，通常のプロセスと区別することが難しい．Flush+Reload 攻撃に対しては，その攻撃過程によるキャッシュ参照などのパターンを機械学習などを用いてモデル化し，攻撃を検知する手法が存在する [17,18]．しかしこうした手法はいずれもキャッシュ参照をパラメータとしたモデル化を行なっており，Flush+Flush 攻撃のようにキャッシュ参照を行わない攻撃は検知できないと考えられる．表 1 に，パフォーマンスカウンタを用いた攻撃検知手法と，代表的な攻撃手法に対する有効性をまとめる．

表 1: パフォーマンスカウンタによる検知手法と有効性
Table 1 Detection methods with performance counters and their effectiveness.

検知手法	Prime+Probe	Flush+Reload	Flush+Flush
PC の総量	✓	✗	✗
モデル化	✓	✓	✗

4. Untrust なプロセスの監視による攻撃検知手法の提案

本論文では，CPU パフォーマンスカウンタを利用し，ステルス性の高い Flush+Reload 攻撃を検知するための手法を提案する．具体的な手法としては，一定時間間隔ごとにキャッシュ参照およびキャッシュミス回数を測定し，Flush+Reload 攻撃特有のパターンを定式化する．

まずはじめに Flush+Reload 攻撃環境の構築および攻撃実験を行い，次に Flush+Reload 攻撃に対する既存の検知手法の有効性を確認する．具体的には，Flush+Reload 攻撃時のキャッシュ参照およびキャッシュミス回数の総量を計測し，他のプロセスと区別できるかを検証する．そして最後に，Flush+Reload 攻撃時のキャッシュ参照およびキャッシュミス回数を 100 msec ごとに測定し，パターンを定式化する．

4.1 Flush+Reload 攻撃環境の構築および実験

4.1.1 攻撃モデル

攻撃モデルは次の通りである．同一の OS 上に攻撃者の攻撃プロセスと，標的となる犠牲プロセスを動かす．犠牲プロセスは GnuPG version 1.4.12 であり，暗号化されたファイルを秘密鍵を使用して復号する．暗号化されたファイルとしては Linux のソースファイルを用いた．GnuPG version 1.4.12 は Square-and-Multiply アルゴリズムを用いている．暗号化あるいは復号化の処理を行う際に，modular exponentiation の計算処理を行う．Square-and-Multiply

```
1: x ⇐ 1
2: for i ⇐ |e| − 1 downto 0 do
3:     x ⇐ x^2
4:     x ⇐ x mod m
5:     if e_i = 1 then
6:         x ⇐ xb
7:         x ⇐ x mod m
8:     end if
9: end for
10: return x
```

図 1: Square-and-Multiply アルゴリズム
Fig. 1 Square-and-Multiply algorithm.

アルゴリズムの場合，公開鍵あるいは秘密鍵の各ビットに対して，**図 1** のように処理を行なっていく．

図 1 のアルゴリズムにおいて，2 行目の $|e|$ は e のビット長を表す．鍵のビットが 0 と 1 の場合で異なる処理を行う．3 行目の処理を square，4 行目の処理を reduce，そして 6 行目の処理を multiply とすると

- ビットが 0 の場合: square, reduce
- ビットが 1 の場合: square, reduce, multiply, reduce

という演算を行う．

攻撃プロセスは犠牲プロセスのいくつかのコードを対象に Flush+Reload 攻撃を行い，犠牲プロセスによるアクセスのパターンを測定し，秘密鍵を復元する．Square-and-Multiply アルゴリズムの場合は，square, reduce, multiply それぞれの関数内のコードのアドレスを対象に，メモリアクセスとキャッシュ階層からのフラッシュ操作を繰り返す．これにより犠牲プロセスによって呼ばれた関数のトレースを行うことができ，結果として使用される鍵のビットの並びを復元することができる．

4.1.2 実験環境

実験環境は，Intel CPU の Xeon X5650 2.67GHz を使用し，OS は Ubuntu 17.10，Linux Kernel 4.13.0 である．メモリは 4GB であり，CPU キャッシュとしては LLC が 12MB サイズである．CPU パフォーマンスカウンタの測定には Linux perf tools version 4.13.16 を用いた．

4.1.3 実験結果

結果として秘密鍵 2048 bit のうち，88 から 90 % のビットを復元することに成功している．

4.2 既存の検知手法の有効性の検証

次に，Flush+Reload 攻撃を行う攻撃プロセスに対して Linux perf tools を使用し，キャッシュ参照およびキャッシュミス回数の総量を測定した．通常のプロセスの測定値と攻撃プロセスの測定値を比較し，Flush+Reload 攻撃を区別できるかを検証する．通常のプロセスとしては Google Chrome によるブラウジング，GnuPG の make そして echo server の挙動を測定した．echo server はクライアント側が一定の周期でデータを送信する場合と，ランダムな間隔でデータを送信する場合の 2 種類の挙動を測定した．

図 3 および**図 4** に測定結果を示す．Prime+Probe 攻撃は Liu らの実装 [11] と Flush+Reload 攻撃の測定結果から推定した値を用いている．

図 3 および図 4 から Prime+Probe 攻撃のキャッシュ参照およびキャッシュミス回数の総量は他のプロセスのそれらと比較して突出して大きくなっており，スレッショルドを定めることで検知が可能であると予想される．その一方で Flush+Reload 攻撃の場合，他のプロセスと比較してキャッシュミス回数は Prime+Probe 攻撃ほど突出して大きくなく，またキャッシュ参照回数に関しては GnuPG の make や Google Chrome によるブラウジングよりも小さくなっていることから，スレッショルドを定めることで Flush+Reload 攻撃を通常のプロセスと区別するのは難しいことがわかる．

4.3 提案手法の有効性の検証

最後に，Flush+Reload 攻撃を行う攻撃プロセスに対して，100 msec ごとにキャッシュ参照およびキャッシュミス回数を測定する．また，Gruss らの論文 [4] を参考に，各時間帯ごとのキャッシュ参照およびキャッシュミス回数はそれぞれ式 (1), (2) のように正規化している．式 (1), (2) において，*cache-references*，*cache-misses* はそれぞれキャッシュ参照回数およびキャッシュミス回数を示し，*iTLB-loads* は Instruction TLB における read/write アクセスによる参照回数であり，*iTLB-load-misses* は *iTLB-loads* におけるミス回数である．

$$kr = \frac{cache\text{-}references}{iTLB\text{-}loads + iTLB\text{-}load\text{-}misses} \tag{1}$$

$$km = \frac{cache\text{-}misses}{iTLB\text{-}loads + iTLB\text{-}load\text{-}misses} \tag{2}$$

図 5 および**図 6** に先ほどの通常のプロセスと Flush+Reload 攻撃の測定結果を示す．図 5 と図 6 により，Flush+Reload 攻撃は通常のプロセスと比較して，キャッシュ参照およびキャッシュミスの回数は双方とも，時間経過によりあまり変化していないことがわかる．これは Flush+Reload 攻撃が常に一定の間隔で，なおかつ同じ分量のアドレスのデータあるいはコードにアクセスを行なっているためである．今回は検証していないものの，Prime+Probe 攻撃においても同様の兆候が見られるため，次の式 (3), (4) を考える．

$$V_{kr} = \frac{1}{n}\sum(kr_i - \overline{kr}_{term})^2 \tag{3}$$

$$V_{km} = \frac{1}{n}\sum(km_i - \overline{km}_{term})^2 \tag{4}$$

式 (3) において，$\overline{kr}_{term}$ は，kr_i が属する時間帯の kr の平均である．時間帯は 1 sec ごとに区切ってあり，km の

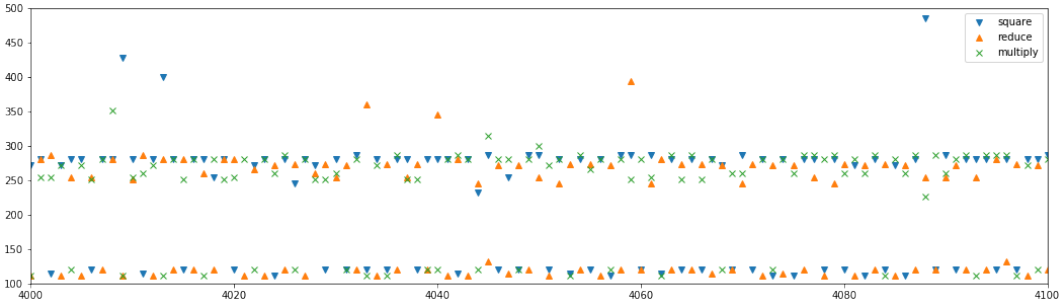

図 2: Flush+Reload 攻撃による犠牲プロセスのアクセスパターン

Fig. 2 An access pattern of a victim process traced by Flush+Reload attack.

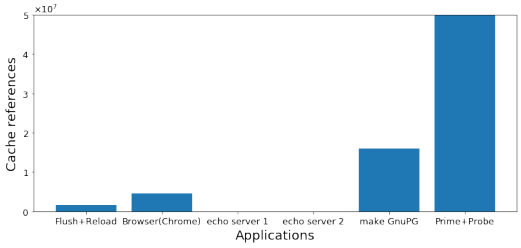

図 3: キャッシュ参照回数の総量

Fig. 3 Total number of cache references.

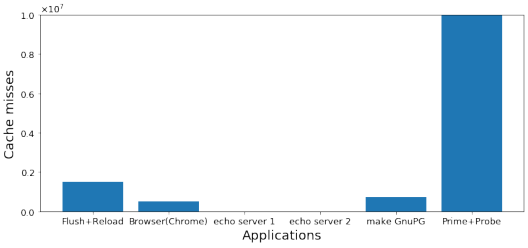

図 4: キャッシュミス回数の総量

Fig. 4 Total number of cache misses.

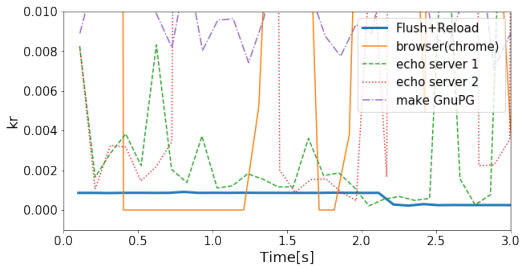

図 5: 100 msec ごとのキャッシュ参照回数の推移

Fig. 5 Change in cache references per 100 msec.

場合も同様である．この式は通常の分散とは異なり，一定時間間隔ごとの分散を表す．通常は短い時間の間に，プロセスによるキャッシュ参照やキャッシュミス回数は大きく変化するため，全区間の平均値を取っても意味をなさない．そこで短い時間間隔ごとに平均をとることで，より微小な変化に対して評価できるようにしている．

式 (3), (4) によって得られた値をプロセスごとに比較したものを**図 7** と**図 8** に示す．図 7 と図 8 を見ると，式 (1) と (2) で算出される V_{kr} および V_{km} の値は，Flush+Reload

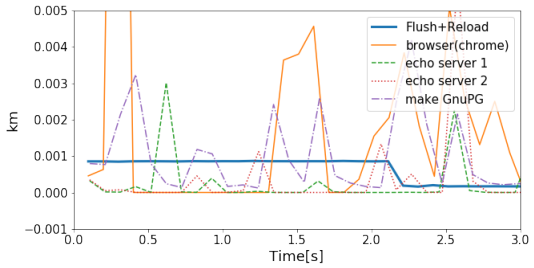

図 6: 100 msec ごとのキャッシュミス回数の推移

Fig. 6 Change in cache misses per 100 msec.

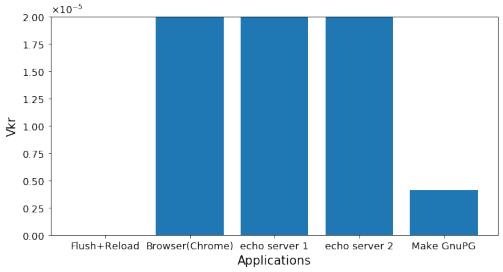

図 7: プロセスごとの V_{kr} の算出値

Fig. 7 Calculated V_{kr} of each process.

攻撃の値がもっとも小さくなっていることがわかる．V_{kr} の場合は，Flush+Reload 攻撃の次に小さい GnuPG の make が Flush+Reload 攻撃の 400 倍程度，V_{km} の場合は echo server 1 が Flush+Reload 攻撃の 30 倍程度となっており，他のプロセスと比較して，Flush+Reload 攻撃の場合は極端に値が小さくなっており，時間経過によるキャッシュ参照回数およびキャッシュミス回数の変化がほとんどないことを示している．そこで，それぞれの分散が次に示す式 (5) のように，定めたスレッショルドよりも共に小さければ Flush+Reload 攻撃であると検知できる．

$$V_{kr} < threshold_{Vkr} \quad \cap \quad V_{km} < threshold_{Vkm} \qquad (5)$$

5. 暗号ソフトウェアのプロセスの監視による攻撃検知の提案

2016 年に提唱された Flush+Flush 攻撃は Flush+Reload 攻撃よりもさらにステルス性の高いキャッシュサイドチャネル攻撃である．Flush+Flush 攻撃は 2.2.3 節で述べた通

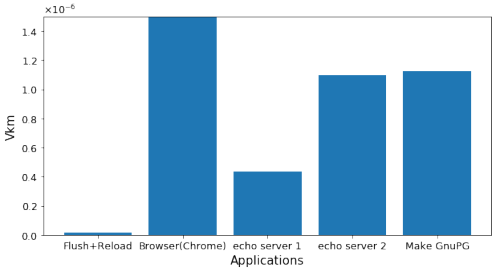

図 8: プロセスごとの V_{km} の算出値

Fig. 8 Calculated V_{km} of each process.

り，その攻撃過程で一切のメモリアクセスを行わない．そのため，攻撃プロセスによるキャッシュ参照やキャッシュミスを測定してもその特徴を得ることはできず，検知モデルを構築することができない．しかし，Flush+Flush 攻撃を受けているプロセスは自身のデータやコードがキャッシュ階層から頻繁にフラッシュされてしまうため，通常時よりも多くのキャッシュミスが発生することが予想される．

そこで本論文では，攻撃を受けている犠牲プロセスのキャッシュミス回数を測定することにより，攻撃を検知する手法を提案する．このような測定により，Flush+Flush 攻撃が検知できることを確認する実験を行なった．犠牲プロセスが攻撃プロセスにより攻撃を受けている場合と，通常のプロセスと犠牲プロセスが同時に動いている場合における，犠牲プロセスのキャッシュミスを測定する．具体的には犠牲プロセスによるキャッシュ参照回数中のキャッシュミス回数の割合を測定する．

犠牲プロセスとしては 4 章と同様，GnuPG version 1.4.12 が暗号化されたファイルの復号処理を行う．実験環境は 4 章と同様である．

具体的なシナリオは以下の通りである．

- GnuPG 単体で動かした場合
- GnuPG に Flush+Reload 攻撃を行なった場合
- GnuPG に Flush+Flush 攻撃を行なった場合
- GnuPG と GnuPG の make 処理が同時に動いている場合
- GnuPG と Google Chrome によるブラウジングが同時に動いている場合
- GnuPG と stress -c 1 が同時に動いている場合
- GnuPG と stress -m 1 が同時に動いている場合
- GnuPG と stress -i 1 が同時に動いている場合

図 9 に測定結果を示す．図 9 はすべてのシナリオについて 10 回ずつ実行し，キャッシュ参照回数に対するキャッシュミス回数の割合の分布を示している．

通常のプロセスと同時に動いている場合は stress -m 1 を除いたすべてのシナリオでキャッシュミスの割合が 15 % 以内に収まっている一方，Flush+Flush 攻撃を受けている場合は 30%，Flush+Reload 攻撃を受けている場合は 50 % にまで上昇していることがわかる．

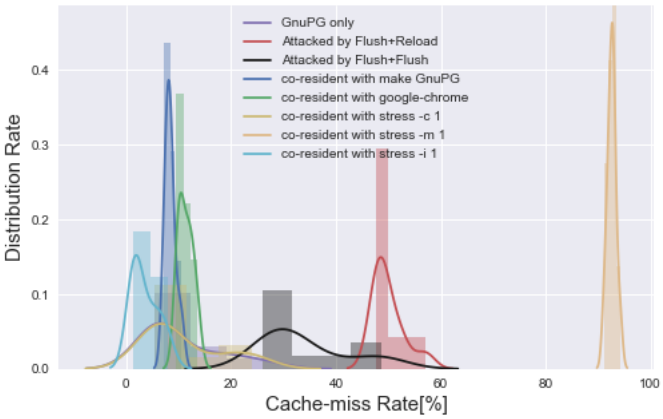

図 9: 各シナリオにおけるキャッシュ参照回数に対するキャッシュミス回数の割合の分布

Fig. 9 Distribution of cache-miss rate of each scenario.

stress -m 1 と同時に動いている場合は 90 % を優に超えるが，stress -m 1 はメモリインテンシブなワークロードであり，メモリアクセスおよびキャッシュ参照回数も極めて多くなる．そのため，犠牲プロセスのコードやデータが頻繁にキャッシュ階層から追い出されてしまい，キャッシュミスが大幅に増加すると考えられる．

今回の実験結果により，モニタリングしている GnuPG が以下の条件を満たした場合に攻撃を受けていると判断することができる．

$$0.25 < \frac{cache\text{-}misses}{cache\text{-}references} < 0.60 \qquad (6)$$

また，攻撃を検知した場合に GnuPG の実行を停止するように設計することで，仮にメモリインテンシブなプロセスが検知範囲に入ったとしても，誤検知による通常のプロセスの強制停止を避けることができる．

6. おわりに

クラウド環境が普及する中で，ユーザは機密情報をクラウド環境でも扱うようになってきた．しかし，キャッシュサイドチャネル攻撃もまた，こうした環境で実行できるように適応してきている．こうした攻撃に対し，これまでにも様々な防御，検知手法が研究されてきた．本論文では，キャッシュサイドチャネル攻撃の中でもステルス性が高く，検知が困難とされてきた Flush+Reload 攻撃，および Flush+Flush 攻撃を検知するために，CPU パフォーマンスカウンタを利用するソフトウェアベースの手法を 2 種類提案した．一つ目は，untrust なプロセスを監視し，Flush+Reload 攻撃の特徴を検知する手法である．二つ目は，暗号ソフトウェアである GnuPG の実行を監視することで，攻撃を受けていることを検知する手法である．

今後は，実験により得られた検知モデルを実際のクラウド環境で適用できるようにシステムとして実装し，パフォーマンスオーバーヘッドや実際にどの程度正確に検知

できるのかを検証していく．

謝辞 本研究は，JST, CREST, JPMJCR1683 の支援を受けたものである．

参考文献

[1] Amazon.com Inc.: Amazon EC2, `https://aws.amazon.com/jp/ec2/`.

[2] Microsoft Corporation: Microsoft Azure, `https://azure.microsoft.com/en-us/`.

[3] Gruss, D., Spreitzer, R. and Mangard, S.: Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches, *24th USENIX Security Symposium (USENIX Security 15)*, USENIX Association, pp. 897–912 (2015).

[4] Gruss, D., Maurice, C., Wagner, K. and Mangard, S.: Flush+Flush: A Fast and Stealthy Cache Attack, *Detection of Intrusions and Malware, and Vulnerability Assessment* (Caballero, J., Zurutuza, U. and Rodríguez, R. J., eds.), Springer International Publishing, pp. 279–299 (2016).

[5] Disselkoen, C., Kohlbrenner, D., Porter, L. and Tullsen, D.: Prime+Abort: A Timer-Free High-Precision L3 Cache Attack using Intel TSX, *26th USENIX Security Symposium (USENIX Security 17)*, USENIX Association, pp. 51–67 (2017).

[6] Percival, C.: Cache Missing for Fun and Profit (2005).

[7] Osvik, D. A., Shamir, A. and Tromer, E.: Cache Attacks and Countermeasures: The Case of AES, *Topics in Cryptology – CT-RSA 2006* (Pointcheval, D., ed.), Springer Berlin Heidelberg, pp. 1–20 (2006).

[8] Gullasch, D., Bangerter, E. and Krenn, S.: Cache Games – Bringing Access-Based Cache Attacks on AES to Practice, *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, IEEE Computer Society, pp. 490–505 (2011).

[9] Zhang, Y., Juels, A., Reiter, M. K. and Ristenpart, T.: Cross-VM side channels and their use to extract private keys, *ACM Conference on Computer and Communications Security* (2012).

[10] Yarom, Y. and Falkner, K.: FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack, *23rd USENIX Security Symposium (USENIX Security 14)*, USENIX Association, pp. 719–732 (2014).

[11] Liu, F., Yarom, Y., Ge, Q., Heiser, G. and Lee, R. B.: Last-Level Cache Side-Channel Attacks Are Practical, *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, IEEE Computer Society, pp. 605–622 (2015).

[12] Xu, Y., Cui, W. and Peinado, M.: Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems, *2015 IEEE Symposium on Security and Privacy*, pp. 640–656 (2015).

[13] Wang, W., Chen, G., Pan, X., Zhang, Y., Wang, X., Bindschaedler, V., Tang, H. and Gunter, C. A.: Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, ACM, pp. 2421–2434 (2017).

[14] Evtyushkin, D., Riley, R., Abu-Ghazaleh, N. C., ECE and Ponomarev, D.: BranchScope: A New Side-Channel Attack on Directional Branch Predictor, *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, ACM, pp. 693–707 (2018).

[15] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M. and Yarom, Y.: Spectre Attacks: Exploiting Speculative Execution, ArXiv e-prints (2018).

[16] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y. and Hamburg, M.: Meltdown: Reading Kernel Memory from User Space, *27th USENIX Security Symposium (USENIX Security 18)*, USENIX Association, pp. 973–990 (2018).

[17] Chiappetta, M., Savas, E. and Yilmaz, C.: Real time detection of cache-based side-channel attacks using hardware performance counters, *IACR Cryptology ePrint Archive*, Vol. 2015, p. 1034 (2015).

[18] Zhang, T., Zhang, Y. and Lee, R. B.: CloudRadar: A Real-time Side-channel Attack Detection System in Clouds (2016).

[19] Pessl, P., Gruss, D., Maurice, C., Schwarz, M. and Mangard, S.: DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks, *25th USENIX Security Symposium (USENIX Security 16)*, USENIX Association, pp. 565–581 (2016).

[20] Razavi, K., Gras, B., Bosman, E., Preneel, B., Giuffrida, C. and Bos, H.: Flip Feng Shui: Hammering a Needle in the Software Stack, *25th USENIX Security Symposium (USENIX Security 16)*, USENIX Association, pp. 1–18 (2016).

[21] Zhang, Y., Juels, A., Reiter, M. K. and Ristenpart, T.: Cross-Tenant Side-Channel Attacks in PaaS Clouds, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, ACM, pp. 990–1003 (2014).

[22] Ristenpart, T., Tromer, E., Shacham, H. and Savage, S.: Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds, *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, ACM, pp. 199–212 (2009).

[23] Liu, F., Ge, Q., Yarom, Y., Mckeen, F., Rozas, C., Heiser, G. and Lee, R. B.: CATalyst: Defeating last-level cache side channel attacks in cloud computing, *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 406–418 (2016).

[24] Kim, T., Peinado, M. and Mainar-Ruiz, G.: STEALTHMEM: System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud, *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, USENIX, pp. 189–204 (2012).

[25] Intel Corporation: Improving Real-Time Performance by Utilizing Cache Allocation Technology, `https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf`.

[26] Bernstein, D. J.: Cache-timing attacks on AES, Technical report (2005).

[27] Vattikonda, B. C., Das, S. and Shacham, H.: Eliminating Fine Grained Timers in Xen, *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, ACM, pp. 41–46 (2011).

[28] Martin, R., Demme, J. and Sethumadhavan, S.: TimeWarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks, *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–129 (2012).