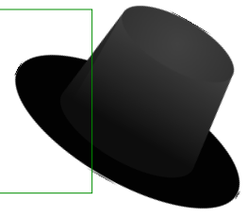
Meltdown & Spectre

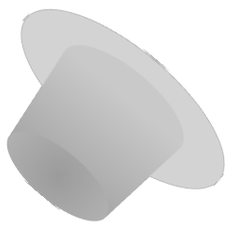
2018/04/15　Version 4.0

seirios

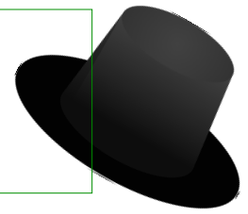
自己紹介



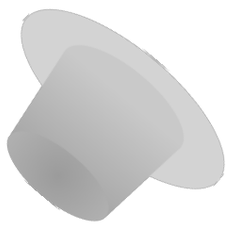
- 略歴
  - 1997年、株式会社インターリンク入社
  - 1999年、株式会社インターネット総合研究所入社
  - 2005年、株式会社IRIコミュニケーションズ転籍（現ブロードバンドセキュリティ）
  - 2010年、株式会社LAC入社
  - 2014年、株式会社ブロードバンドタワー入社
  - 2017年、株式会社IoTスクエア転籍
- 活動
  - v6推進協議会 IPv4枯渇TF 教育・テストベッドWG メンバー（2009～現任）
  - v6推進協議会 セキュリティWG メンバー（2010～現任）



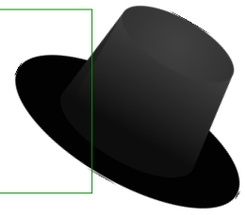
Meltdown と Spectre



- 2018年1月3日、CPUの脆弱性であるMeltdownとSpectreが公開される
  - 報告者は Google Project ZERO (Googleの専門チーム)
  - ゼロデイ攻撃撲滅が「目標」
- CVE-2017-5715 : Spectre Variant 2 / Branch Target Injection
- CVE-2017-5753 : Spectre Variant 1 / Bounds Check Bypass
- CVE-2017-5754 : Meltdown / Rogue Data Cache Load

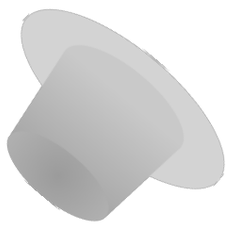


影響範囲は？

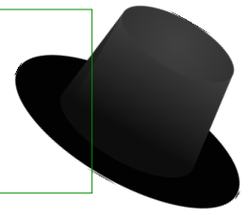


- Meltdown
  - Out of Order execution / Speculative Execution(投機的実行) を実装しているIntel製プロセッサーに潜在的に影響がある
  - 1995年以降のプロセッサー全てつまり、Pentium PRO以降のCPUが影響を受ける可能性があるとIntelが公表
    - ただし、Itanium/2013年以前のAtomを除く
  - 当初はIntelだけと言われていたが、ARMにも存在していることが判明
    - ARM系Andoroid端末にも影響が...
  - IBM POWERファミリーも影響を受けることが判明...

- Spector
  - Branch Prediction(分岐予測) / Speculative Execution(投機的実行) を実装しているプロセッサーに潜在的に影響がある
    - 現代のPiplining を利用するほぼ全てのCPUで影響がある
    - 投機的実行や分岐予測は高速化の重要な技法

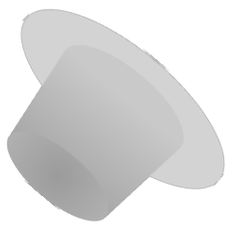


実際のところどうなの？

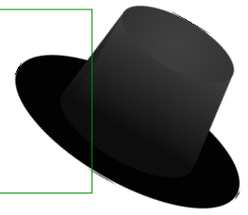


- この脆弱性群は、CPUの「OoOE」と「投機的実行」が鍵
  - 投機的実行に関しては後述
  - Meltdown対応パッチを適用すると、かなり性能が劣化する

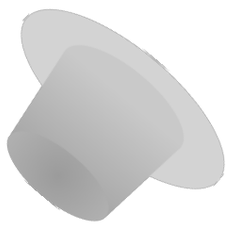
- **「劣化ですってよ、奥さん」**
  - 単なる演算処理能力は劣化しない
  - コンテクストスイッチの速度低下
    - システムコール・ハードウェア割り込みのオーバーヘッドが大きくなる



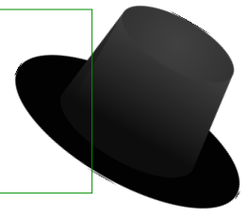
し・く・み ❤️



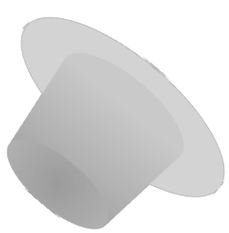
- CPUの内部キャッシュを利用した、サイドチャネルアタックによる不正なデータ読み出し
- ソフトウェアではなくCPUの高速化機能にまつわる脆弱性
  - つまり、単純なSoftware Debugでは対応できない
- 前提となる知識
  - Side Channel Attack
  - Flush+Reload Attack
  - Out of Order Execution
  - Branch Prediction
- Meltdown/Spectreによって、コード上は（論理的には）アクセスできない（はずの）データが**読み出し可能に**



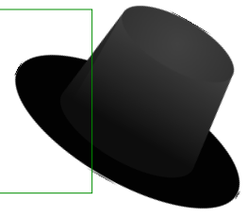
CPUに利用されているアーキテクチャ



- **Instruction Pipeline**(命令パイプライン)
  - 複数の命令を同時にfetchし、複数の実行ユニットを並列に動作させる性能向上手法
- **Out of Order Execution**
  - 高性能プロセッサにおけるクロックあたりの命令実行数を増やすための手法
  - プログラム中の命令の並び順によらず、処理可能な命令について、逐次開始・実行・完了させる。対比語としてIn Order実行がある
- **Speculative Execution**(投機的実行)
  - 性能最適化の一種。予定されている処理を行うかどうかを確定させる前に予定処理を行い、処理時間を稼ぐ。代わりに計算資源（パイプラインなど）に余裕がなければならない
  - PreFetch(Filesystem/memory)や分岐予測(CPU)、楽観的並行性制御(DB)で利用されることが多い



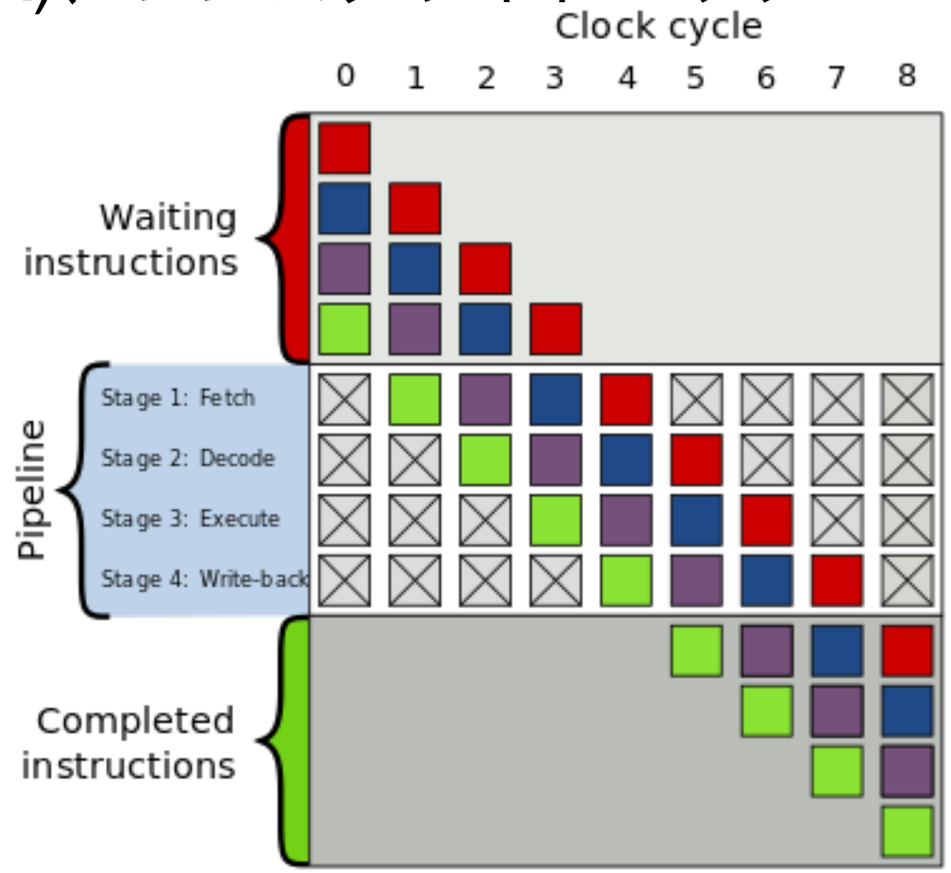
Instruction Pipeline



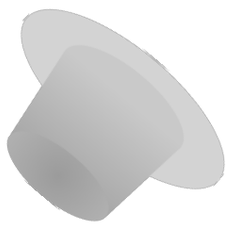
- Instruction Pipeline(以下IP)は、命令スループットを向上させる設計技法のひとつ
  - IPを持つプロセッサは、命令の処理にあたって、細分化された「独立して実行できる工程」に分割する。
  - 分割された各工程を並列化して全体の処理時間を大幅に軽減する
  - 命令fetch(IF)、命令Decode(ID)、Execute(EX)、レジスタライトバック(WB)などのように内部処理を細分化する

右図が４段パイプラインの例。

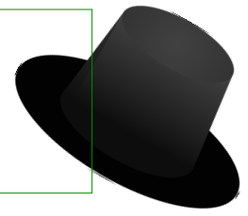
各命令が細分化されて流れていくのがわかる。



WikiPediaより



In Order実行とOut of Order実行



In Order

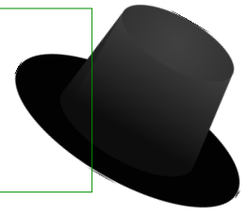
1. 命令フェッチ
2. オペランド（データ）準備
3. 実行ユニットへの割当
4. 実行
5. 結果をレジスタに格納

- 現代のCPUでは、
  CPU内キャッシュ<<メモリー内データ<<I/Oの向こうにあるデータ(DiskやNetworkなど)
  の順にデータ受け取り時間が**激烈に増加**する
- パイプラインが多段になればなるほど、**無駄なパイプラインを作らない**ことが必要になる。
- CPU内のパイプラインを遊ばせないために、**処理できるものから処理**するというのがOut Of Order実行の肝

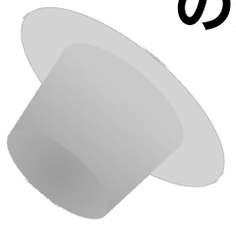
Out of Order

1. 命令フェッチ
2. 命令にバッファエントリーを割当
3. 命令を待ち行列から実行キューに送る
4. 待ち行列内命令はオペランドを待つ
5. オペランドを受け取って実行キューへ
6. キュー内の命令に実行ユニットを割当
7. 実行
8. 結果をバッファに格納
9. バッファ内の最も古い命令の実行終了を待つ
10. 古い命令から実行結果をレジスタに格納

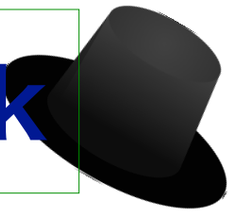
投機的実行



- 近年のCPUは、性能向上のために、多段パイプライン構造を採用
- プログラムは「**条件分岐が必須**」
- 条件分岐は、条件判断の結果が出るまで**行き先を決定できない**
  - すなわち、OoO実行することができず、待ちが増え、結果性能が落ちる
- 投機的実行は、判断結果を過去の実行履歴に基づいて**先行予測(Branch Prediction)** することで、確率的に**実行される可能性が高い方の処理を先に実行**しておくこと。結果、予測が間違っていた場合、そこまでの結果を捨ててやり直し。
- 積極的実行は、投機的実行の一種であり、**「両方の経路を実行」**し、**条件分岐判断の結果をみて採用する結果を決める**手法

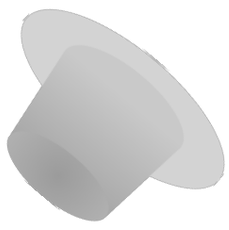


Side Channel AttackとFlush+Reload Attack

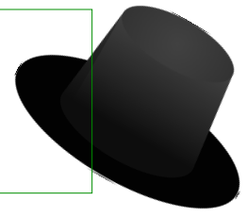


- Side Channel Attack
  - 暗号化されたデータ（等）を（直接データにアクセスせずに、他のプロセスなどが）当該データを処理している際に発生する**物理的な現象を観測することによって推定する**攻撃手法
    - 処理時間の違い、消費電力の違い、電磁波の違いなどが有名

- Flush+Reload Attack
  - キャッシュメモリーを利用したSide Channel Attackの一種
    - 初期化、抜き取り、復元を利用する
      - Cache Region (領域) をPurge（初期化）
      - 必要な情報を読み込み、Cacheに載せる
      - **On cache**と**Not on cache**の際のCacheのLookup速度差を利用して復元



Spectre 手法(Variant 1)



問題のあるコードの例

```
char a[100];
int len = 100;

void foo(x) {
  int ret = 1;
  if (x < len) {     ←(A)
    ret = probe[ a[x] * PAGE_SIZE ];  ←(B)
  }
  return ret;
}
```

a[x] は aがNバイト型(ここではcharなので1byte)の配列なので、

”a+x*N”で計算されるアドレス

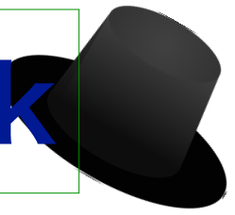
の値をロードしたもの

ここで、**x = (z - a)/N** となるxを選ぶと、**アドレスzの値を読み出す**コードを**投機的に実行**できる

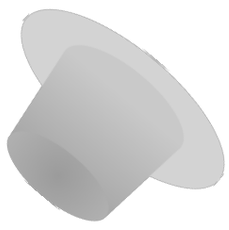
なお、PAGE_SIZEはCache lineのサイズより大きければ良い

1. OoO Executionによって以下の２つの流れが並列に実行される
   A. lenをload
      → (x < len)を評価
      → 分岐
   B. a[x]をload
      → probe[a[x]*PAGE_SIZE] をload
      → retに probe[...]をストア
2. ここで、foo(1000)を呼ぶと...
   - 流れBが先行する
     A. lenをload
     B. a[x]を....
   - Aで分岐予測失敗を検出する
     A. 評価した結果、Falseになる
     B. retにprobe[...]をストア中
3. 流れAが終了し、流れBの実行結果は捨てる
   - 論理的には読み込んでいないはずのa[1000]がprobe[...]から復元可能

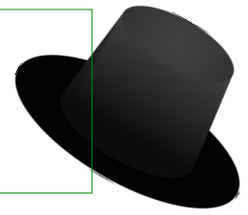
Variant 1を利用したFlush+Reload Attack



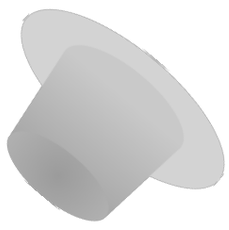
- できることは何か？
  - fooの引数変更によって、攻撃対象プロセスがアクセス可能な任意のメモリーを読み出せる
  - kernel内で実行することができれば、kernel内のメモリーが読み出せる
- 処理の流れ
  1. 初期化: probe[]を初期化し、cacheをクリアする
  2. 学習: foo()を何度も呼び出して、if文の分岐予測先をTRUEにする
  3. 抜取: foo()を実行、if文の分岐予測を失敗させ、データをprobe[]に抜き出す
  4. 復元: probe[]内の情報を復元する
- 現実的な攻撃方法
  - 攻撃対象バイナリに（元々存在するI動的生成した）コードを利用して、他の方法と組み合わせて攻撃する　→　**現実にはなかなか大変**



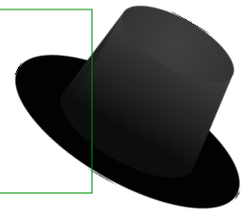
Variant 2概要



- Variant 1では、**条件分岐**を利用して**投機実行**させた
  - トレーニングさせるのは条件分岐のみで、**Jump先は予め決定**されている
- Variant 2では、**Indirect Branch**命令を利用する
  - Indirect Branch → 実行時に初めてJump先が判明する命令
    - C/C++の場合、関数ポインタや仮想関数呼び出し等が該当
    - Indirect Branchでも**過去の履歴に基づいた投機実行**が行われるのを利用し、Jump先のコードの投機実行を引き起こし、Flash/Reloadで変化検出する
      - J**ump先そのものをトレーニング**できる



Variant 2 - In Process (1) -



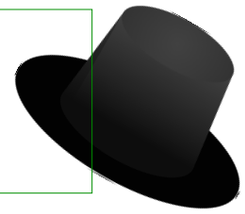
Core Code(C/Assembler)

```
indirect_call:
    mov rax,rcx
    mov rcx,rdx     ; Preparing 1st Parameter
    mov rdx,r8      ; Preparing 2nd Parameter
    clflash [rax]
    call [rax]
    ret

void do_nothing(uint8_t*, uint8_t*) {}

touch_and_break:
    movxz eax, byte [rcx]
    shl rax, 0ch
    mov al,byte [rax+rdx]
    sysenter
```

出典: https://qiita.com/msmania/items/eab61240f8b4f71b0177#spectre-variant-2-branch-target-injection-cve-2017-5715

- Indirect_call
  - Jump先のアドレスが格納されたポインターをrcxで受け取る
  - jump先の格納アドレスをclflash
    - 投機実行の時間稼ぎ
  - callでindirect jumpを実行
- do_nothing: 何もしない
- touch_and_break
  - 投機実行させたい関数
  - flash/reloadの仕込みをする
  - systemcallを実行
    - 普通に呼び出すと、systemcallの設定を何もしていないため、例外が発生する

Variant 2 - In Process (2) -



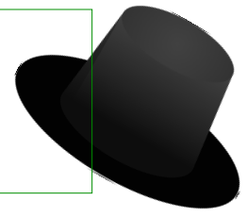
PoC

```
uint8_t train_and_attack[6] = {};
train_and_attack[5] = 1;

for (auto x : train_and_attack) {

  *reinterpret_cast<uint8_t*> (touch_and_break)
    = x ? original_prologue : 0xc3;

  target_proc =
      x ? do_nothing : touch_and_break;

  indirect_call( call_destination, target_address,
probe );
```

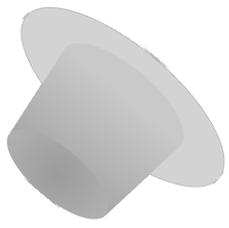
出典: https://qiita.com/msmania/items/eab61240f8b4f71b0177#spectre-variant-2-branch-target-injection-cve-2017-5715

- 上記コードを利用し、トレーニング（5回程度）を実施する
  - Indirect_callの第一引数にtouch_and_breakのアドレスが入ったポインタを渡す
    - ただし、単純にtouch_and_breakを呼ぶと末尾のsysenterでProcessがCrash
  - そのため、touch_and_breakの先頭にret命令(0xc3)を埋めて、呼ばれたら即関数が終了するようにする
- 本番
  - touch_and_breakの先頭を元に戻す
  - indirect_callの第一引数にdo_nothingのアドレスが入ったポインタを渡す

Variant 2 - Inter Process (1) -

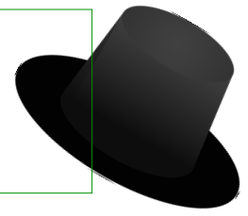


- Variant 2が真に強力なのは、別プロセスに働きかけることができること
  - CPU内で、Indirect Branchを管理しているBranch Predictorがプロセスで分離されていないため
    - 比較的昔から推測されていたアイデアではある
  - このアイデアを利用して、トレーニングと本番を別プロセスで実行できる
- 成立させるための基本条件 (GPZの論文より)
  - Jump先のメモリーページが実行可能であること
  - プロセス間で共有されるモジュールのメモリー領域はどのプロセスからでもclflash可能である
  - Jump履歴の管理は、Jump元仮想アドレスの全ビット「ではなく」一部の下位ビットが利用される
  - Branch PredictorはJump先の仮想アドレスに着目するが、命令コード・物理アドレス・プロセスIDは考慮しない
  - Branch Predictorは CPUコア間で独立しており、トレーニングは異なるコアには影響を与えない
    - 但し、出典元の実験では、CPUコア単位での独立ではなくHyper Thread単位での独立である可能性もある
  - Windowsにおいて、ロードされた実行可能モジュール(EXE/DLL)は、基本的に「複数のプロセス」から共有され、「共通の仮想アドレス」が利用される



この基本条件は大変に厳しいので、現実にVariant 2で攻撃することは難しい

Variant 2 - Inter Process (2) -



- 論文によると
  1. 間接分岐実行時に「外部から値を制御できる」レジスターが存在する箇所を選ぶ
     - 攻撃者が何らかのデータを入力できる必要がある。頑張って探す
  2. 制御可能なレジスターを使うと任意の箇所をFlash/Reloadできるコードを共有DLLの実行可能領域から探す(以下gadget)
     - 頑張って見つけるしかない。論文では、ntdll.dllを利用している模様
  3. 攻撃の実行

     3.1.攻: Probe領域のclflush

     3.2.攻: gadgetへの分岐を行うようにトレーニング実行

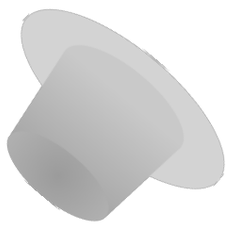
     3.3.攻: 被攻撃プロセスが保持している間接分岐領域のキャッシュクリア

     3.4.被: gadgetへの間接分岐の実行

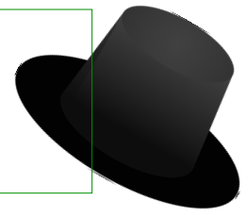
     3.5.攻: Probe領域の探索

3.3と3.4のタイミングを合わせることは至難であり、これができるかが V2 成功の鍵

→ つまり、非常に攻撃を成功させることが難しい

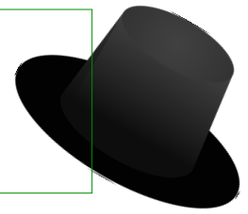


Meltdown 前提知識



- 問題の根本
  - 投機的実行において、**プロセスを分離していない、かつ、Kernelメモリー内にある「低特権データ」と「高特権データ」を区別していない**
    - この結果、悪意あるプログラムが投機実行において「特権のないコード」を用いてKernelメモリーにアクセスすることが可能になる
- 既存のOSにおいては、VM(Virtual Memory ≠ Virtual Machine)を利用して、各プロセスのメモリー空間を管理している
- 各プロセスのVM空間に全PM(Physical Memory)を配置している
  - 実装の単純化、高速化のため
- 従って、OSは以下の制約をプロセスに対して課さなければならない
  - Process(Proc)からProcに割り当てられたメモリー空間へのアクセス　→　**許可**
  - **ProcからKernel空間へのアクセス**　→　**禁止**
  - **Syscallを介した**ProcからKernel空間へのアクセス　→　**許可**

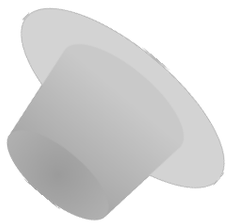
Meltdown 手法



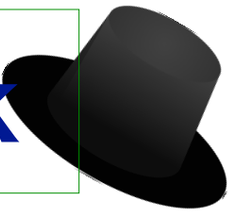
問題のあるコードの例

```
char *p = SOME_KERNEL_ADDRESS;
char *probe[ 256 * PAGE_SIZE ];

Do_Not_Use = probe[ (*p) * PAGE_SIZE ];
```

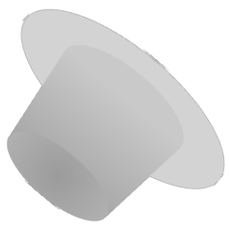
1. 内部的に以下の2つの流れが並列実行される
    A. pへのアクセス権限チェック
        → 無権限ならBの実行結果を捨てる
    B. (*p)をload
        → probe[(*p) * PAGE_SIZE]をload
2. 流れBが先に完了する
    A. pへのアクセス権限チェック中
    B. (*p)をload
        → probe[(*p) * PAGE_SIZE]をload
3. 流れAが終了
    A. 無権限が判明
        → Bの実行結果を捨てる
        → 例外発生
    B. Aの完了待ち
4. 論理的に読み込んでいないはずの(*p)をprobe[]から復元可能



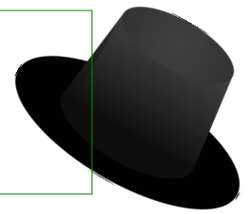
Meltdownを利用したFlush+Reload Attack



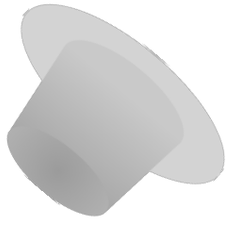
- Main Routine
  1. SIGSEGVのSignal Handlerを登録する
  2. setjump()。登録時は処理3へ。longjump()による復帰時は処理5へ
  3. 初期化: probe[]を初期化し、cacheをクリアする
  4. 抜取: SIGSEGV発生
  5. 復元
- SIGSEGV Handler
  - longjump()でMain Routineの処理2に飛ぶ
- 現実的な攻撃方法
  - 攻撃対象システムから一般ユーザー権限でプロセスを実行できればOK
  - 任意の物理メモリーを読み出せる　→　ファイルの内容・他のユーザーのメモリー
    - **Container貸しやマルチテナントサービスシステムはとんでもなくヤバイ**



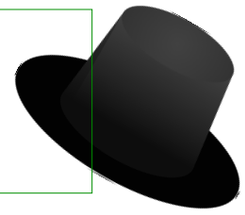
対策：Meltdown



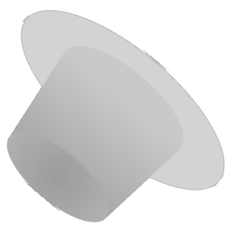
- Kernel Page Table Isolation(KPTI)
  - Isolation: 分離・隔離
- KPTI とは...
  - プロセスのアドレス空間に kernel memoryをマップしない
    - Linux 4.15に取り込まれた。各種DistroにBack portされた
      - OpenBSD/FreeBSDにも取り込まれた
        - NetBSDはCurrentには取り込まれているがReleaseはまだの模様
      - **kernelメモリーがマップされない**ので、さわりようがない
      - 通常はマップされないが、syscallが発行されると、syscall先でmapする
- PCIDによるKPTIの緩和
  - PCID(Process Context IDentifier)
    - SandyBridgeから追加...Haswell以前のCPUにはINVPCID命令がないので役に立たない



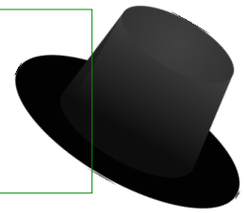
対策：Spectre



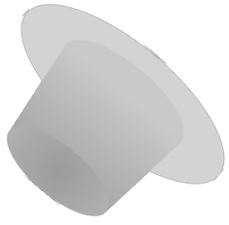
- Variant 1への対策
  - Linux: eBPFを無効化するなど
  - Compilerレベルでの対策（配列境界チェックを埋め込むなど）
- Variant 2への対策
  - 分岐予測の無効化→大きな性能劣化を伴う
  - Retpoline: 間接Jump命令を書き換え、間接分岐先を予め制御する
    - Intel Processorにおいては、通常の間接Jumpと関数呼び出しからの戻りでの間接ジャンプでは、分岐予測バッファが異なることを利用する
    - 通常の間接Jump命令を、関数呼び出しとスタックの戻り先の書き換えによって実現し、分岐予測先を騙して投機実行を止める



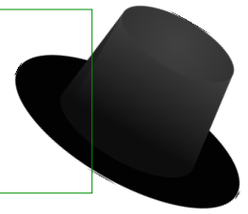
対策：では、Intelは？



- 影響のあるCPUに対し、Microcode updateを提供すると発表
  - 後日、一部CPUは対策供給しないと発表
- Microcode updateとは？
  - 発売後のCPUに対するUpdate
  - 多くはBIOSを更新してCPUに新たなMicrocodeを送り込む
- 何を追加したの?
  - Variant 2対策の投入
    - IBRS　　Indirect Branch Restricted Specuration
      - 間接分岐命令の投機実行を行う際に、IBRSが1にセットされる前の分岐予測バッファの結果を利用しないようにする（Kernel空間→User空間遷移時に0にする）
    - STIBP　　Single Thread Indirect Branch Predictors
      - (一部のCPUにおける) Hyper-Threading環境での、論理CPU間での分岐予測バッファの共有をやめる
    - IBPB　　Indirect Branch Predictor Barrier
      - 分岐予測バッファをプロセスやVMの切り替え時などにフラッシュし、分岐予測の状態を初期化する



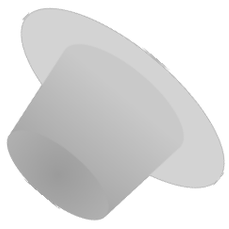
KPTIの有無



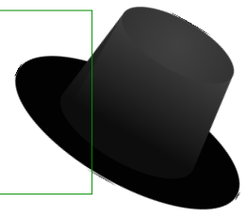
KPTI なし	KPTI あり	PCIDを用いたKPTI緩和
User → kernel への遷移 : syscall発行時など		
ページテーブルを**切り替えない**	ページテーブルを**切り替える**	ページテーブルを**切り替える**
TLBは**フラッシュしない**	TLBは**フラッシュする**	TLBは**フラッシュしない**
Context Switch		
ページテーブルを切り替える	ページテーブルを切り替える	ページテーブルを切り替える
TLBは**フラッシュしない**	TLBは**フラッシュする**	TLBは**フラッシュしない**
kernel → User への遷移		
ページテーブルを**切り替えない**	ページテーブルを**切り替える**	ページテーブルを**切り替える**
CSしたら**User空間のTLB**をフラッシュ	CSしたら**全TLB**をフラッシュ	CSしたら**切り替え前のプロセス用TLBをフラッシュ**

CS: Context Switch

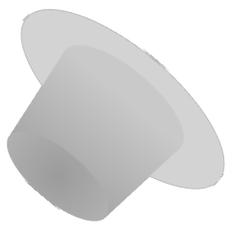
TLB: Translation Lookaside Buffer : CPUが仮想アドレスと物理アドレスを対応(map)させた情報を一時的に保管しておくバッファ



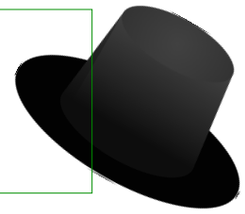
性能劣化



- Page Table切り替え増加 = vmstatにおける **sys** が増加
- TLBフラッシュ増加 = TLBミス増加
  - User空間で発生した場合 **user** が増加、kernel空間で発生した場合 **sys** が増加
- その他諸々の追加書実行にかかるコスト = **sys** が増加
- 一連の処理によるCache使用料増加 = Cache miss増加
  - User空間で発生した場合 **user** が増加、kernel空間で発生した場合 **sys** が増加
- Intelによれば、「新しいCPUほど影響が小さい」
  - まぁ、それはそうだろうなぁ。

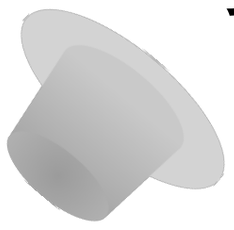


なおIntelの発表

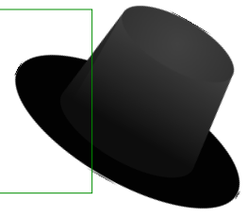


- Meltdown : Rogue Data Cache Load の解決方法
  - ユーザモードとスーパバイザモードのページ構造を分け、分離することで解決できる、はず
    - ソフトウェアによる修正が必要ということ = Page Table Isolation
- Spectre Variant 1: Bound check Bypass の解決方法
  - 基本的にはソフトウェアによる解決法が望まれているもよう。
  - 実際にアクセスを行ってはいけないシーケンスについては、FENSE命令を使って明示的にメモリアクセスの同期を行い、ハードウェアプリフェッチを防ぐ
- Spectre Variant 2: Branch Target Injection の解決方法
  - プロセッサとシステムソフトウェアの間に新しいインタフェースを仕込む、らしい。
    - Indirect Branch Restricted Speculation (IBRS): 間接分岐の投機実行を制限する。
    - Single Thread Indirect Branch Predictors(STIBP): 間接分岐の投機的移行は、1つのスレッドのみが実行できるように制約する。
    - Indirect Branch Predictor Barrier(IBPB): 前のプログラムの挙動が、間接分岐の分岐予測に影響しないようにする。

要するに、CPUの高速化手法を制限することになる＝性能は落ちる



なお2/20発表



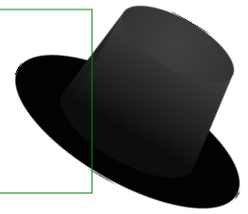
ZDNet Japan > セキュリティ

CPU脆弱性「Meltdown/Spectre」を悪用した新たな攻撃手法--研究者らが発見

出典：https://japan.zdnet.com/article/35114939/

- The Registerが元記事を公開した
- Princeton大学の研究者２名とNVIDIAの人が論文を公開
  - https://arxiv.org/pdf/1802.03802.pdf
  - MeltdownPrime / SpectrePrime と命名したらしい。(Amazonか！)
    - 暇がなくて読めてません
  - Prime+Probeという手法を利用したらしい
    - http://palms.ee.princeton.edu/system/files/SP_vfinal.pdf
    - 何れにしてもCache Timing Attackの一種
- 攻撃方法の違い
  - 新種の攻撃方法はCPU２つを相対させていること
  - CPUのメモリーキャッシュを利用し、アプリを実行しながら重要情報を読み取る
- おそらく、Meltdown Patch当てれば、緩和できると予想できる（？）

なお、2/21発表



OpenBSD Journal

Home | Archives | About | Submit Story | Create Account | Login

Meltdown fix committed by guenther@

Contributed by Paul 'WEiRD' de Weerd on 2018-02-21 from the so-hot-of-the-press-it-melts-your-cpu dept.

Meltdown mitigation is coming to OpenBSD. Philip Guenther (guenther@) has just committed a diff that implements a new mitigation technique to OpenBSD: Separation of page tables for kernel and userland. This fixes the Meltdown problems that affect most CPUs from Intel. Both Philip and Mike Larkin (mlarkin@) spent a lot of time implementing this solution, talking to various people from other projects on best approaches.

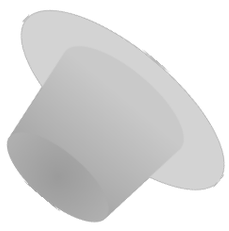
In the commit message, Philip briefly describes the implementation:

```
Meltdown: implement user/kernel page table separation.

On Intel CPUs which speculate past user/supervisor page permission checks,
use a separate page table for userspace with only the minimum of kernel code
and data required for the transitions to/from the kernel (still marked as
supervisor-only, of course):
- the IDT (RO)
- three pages of kernel text in the .kutext section for interrupt, trap,
and syscall trampoline code (RX)
- one page of kernel data in the .kudata section for TLB flush IPIs (RW)
- the lapic page (RW, uncachable)
- per CPU: one page for the TSS+GDT (RO) and one page for trampoline
stacks (RW)

When a syscall, trap, or interrupt takes a CPU from userspace to kernel the
trampoline code switches page tables, switches stacks to the thread's real
kernel stack, then copies over the necessary bits from the trampoline stack.
On return to userspace the opposite occurs: recreate the iretq frame on the
trampoline stack, switch stack, switch page tables, and return to userspace.

mlarkin@ implemented the pmap bits and did 90% of the debugging, diagnosing
issues on MP in particular, and drove the final push to completion.
Many rounds of testing by naddy@, sthen@, and others
Thanks to Alex Wilson from Joyent for early discussions about trampolines
and their data requirements.
Per-CPU page layout mostly inspired by DragonFlyBSD.
```

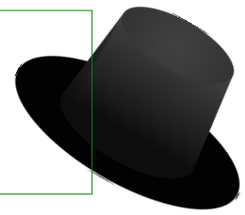
Even with extensive testing by developers, editors@ are sure more testing would be welcome. So grab a snapshot (if they're from February 22nd or later, they should contain the diff) and see how this behaves in your environment!

@2018/02/21 https://undeadly.org/cgi?action=article;sid=20180221201856

- OpenBSD用Meltdown対策Patch
  - まだ、Releaseされたわけではない
  - 対応策は、KPIS (Kernel Page Table Separation) らしい
  - KPTIと概ね同手法
    - ここにはOBSDユーザーいないよね...
  - Per-CPU page layoutに関してDragonfly BSDを参考にしたらしいことが書かれている



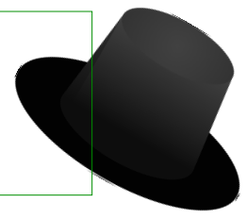
なお、3/14



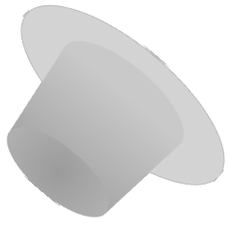
- FreeBSD-SA-18:03.speculative_execution
  - Meltdown対策 (KPTI)が取り込まれる
    - sysctl vm.pmai.pti パラメータでPTIのOn/Offが設定できる
      - defaultは1で、PKTI有効
  - Spector V2対策が取り込まれる
    - sysctl hw.ibrs_disable パラメータでCPUのIBRS利用のOn/Offが設定できる
      - defaultは1でIBRSを利用しない
      - MicrocodeをUpdateしないとIBRSは利用できないため

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

=============================================================================
FreeBSD-SA-18:03.speculative_execution                      Security Advisory
                                                          The FreeBSD Project

Topic:          Speculative Execution Vulnerabilities

Category:       core
Module:         kernel
Announced:      2018-03-14
Credits:        Jann Horn (Google Project Zero); Werner Haas, Thomas
                Prescher (Cyberus Technology); Daniel Gruss, Moritz Lipp,
                Stefan Mangard, Michael Schwarz (Graz University of
                Technology); Paul Kocher; Daniel Genkin (University of
                Pennsylvania and University of Maryland), Mike Hamburg
                (Rambus); Yuval Yarom (University of Adelaide and Data6)
Affects:        All supported versions of FreeBSD.
Corrected:      2018-02-17 18:00:01 UTC (stable/11, 11.1-STABLE)
                2018-03-14 04:00:00 UTC (releng/11.1, 11.1-RELEASE-p8)
CVE Name:       CVE-2017-5715, CVE-2017-5754

Special Note:   Speculative execution vulnerability mitigation is a work
                in progress.  This advisory addresses the most significant
                issues for FreeBSD 11.1 on amd64 CPUs.  We expect to update
                this advisory to include 10.x for amd64 CPUs.  Future FreeBSD
                releases will address this issue on i386 and other CPUs.
                freebsd-update will include changes on i386 as part of this
                update due to common code changes shared between amd64 and
                i386, however it contains no functional changes for i386 (in
                particular, it does not mitigate the issue on i386).

For general information regarding FreeBSD Security Advisories,
including descriptions of the fields above, security branches, and the
following sections, please visit <URL:https://security.FreeBSD.org/>.

I.   Background
```

なお4/2



- Intelが一部の古いCPUに対するSpectre対策を中止
  - IntelのMicrocode Revision Guidanceによる
- 中止した理由
  - マイクロアーキテクチャの特性上、「Variant 2」を緩和する機能の現実的な実装が不可能
  - 提供されている商用システムソフトウェアサポートが限られている
  - (Intelの)顧客からの情報によると、これらの製品のほとんどは「クローズドシステム」として実装されており、この脆弱性を突かれる可能性が低いと見られる
- UpdateされないCPU Family
  - x86/arm64系
    - Bloomfield　Clarksfield　Gulftown　Penryn/QC
    - Harpertown C0　Harpertown E0　Jasper Forest　Yorkfield
    - Wolfdale C0　Wolfdale M0　Wolfdale E0　Wolfdale R0
  - ATOM系
    - SoFIA 3GR



お疲れ様でした