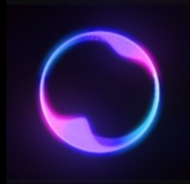
10. SGX攻撃編②

Ao Sakurai

2024年度セキュリティキャンプ全国大会
S3 – TEEビルド&スクラップゼミ

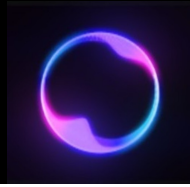
本セクションの目標



- Controlled-Channel攻撃とÆPIC Leak攻撃、そしてSGXに対する過渡的実行攻撃の1つであるForeshadow攻撃の解説を行う
- mprotectシステムコールを用いる事により、ごく簡単かつ擬似的なControlled-Channel攻撃を実践する
- SGX-Vaultの実装の一部に対し、Controlled-Channel攻撃に対する軽減策の導入を行う

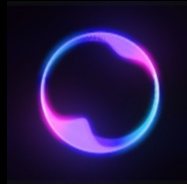
Controlled-Channel攻撃

Controlled-Channel攻撃

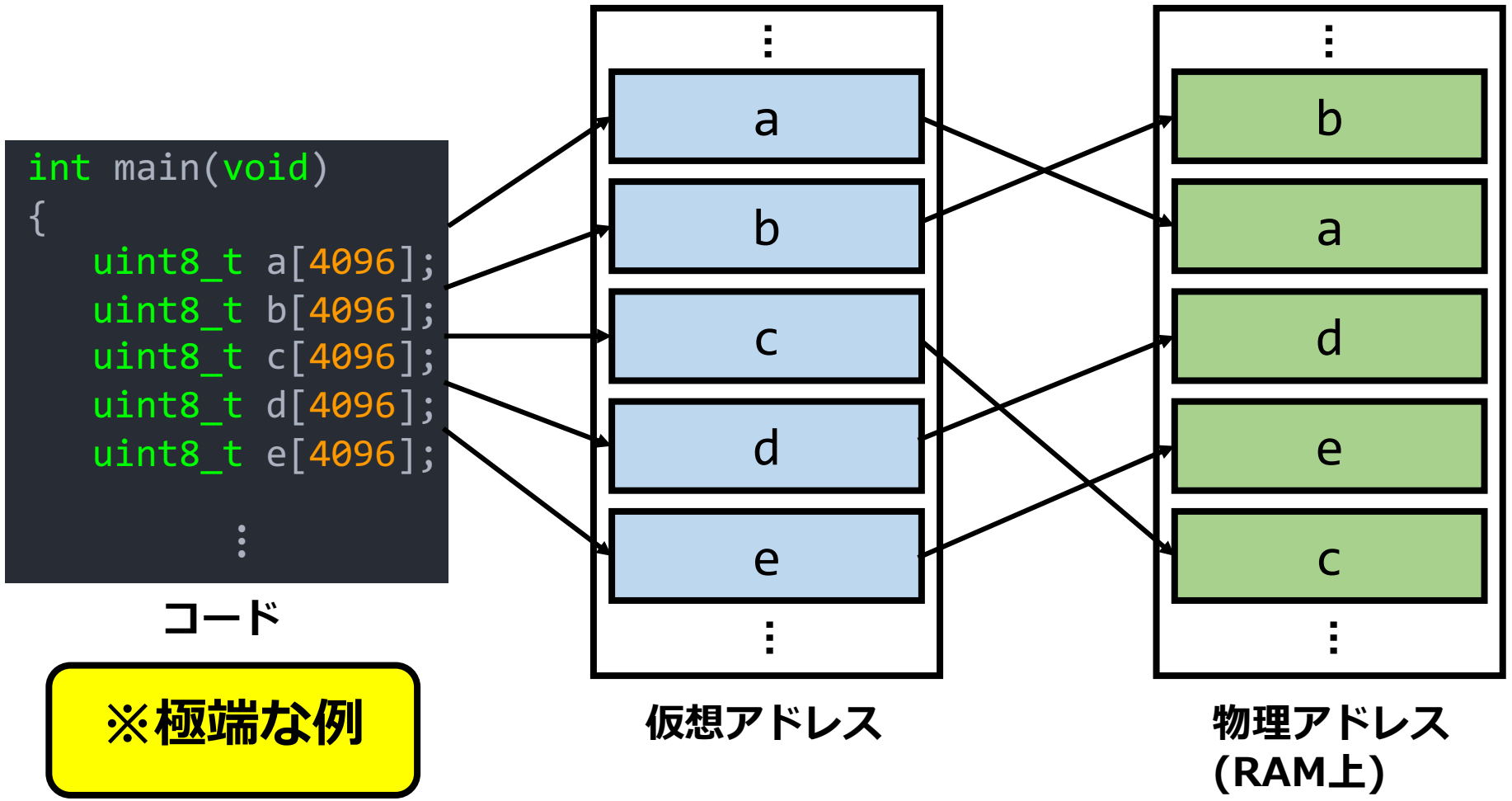


- **SGX**のような**防護システム**に対して**非常に有効**な**サイドチャネル攻撃**の一つ
- この攻撃では**ページフォルト**を悪用する
- 以下、OSは**攻撃者**により**制御権を掌握されている**と仮定する
- この攻撃では**3つ**のフェーズが存在する：
  1. コード・実行バイナリに対する**オフライン解析**
  2. **オンライン解析** (①実行バイナリを実行; ②ページフォルト起動; ③観測)
  3. 観測結果から**秘密を推定**

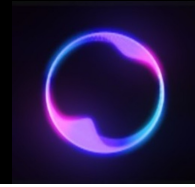
ページング



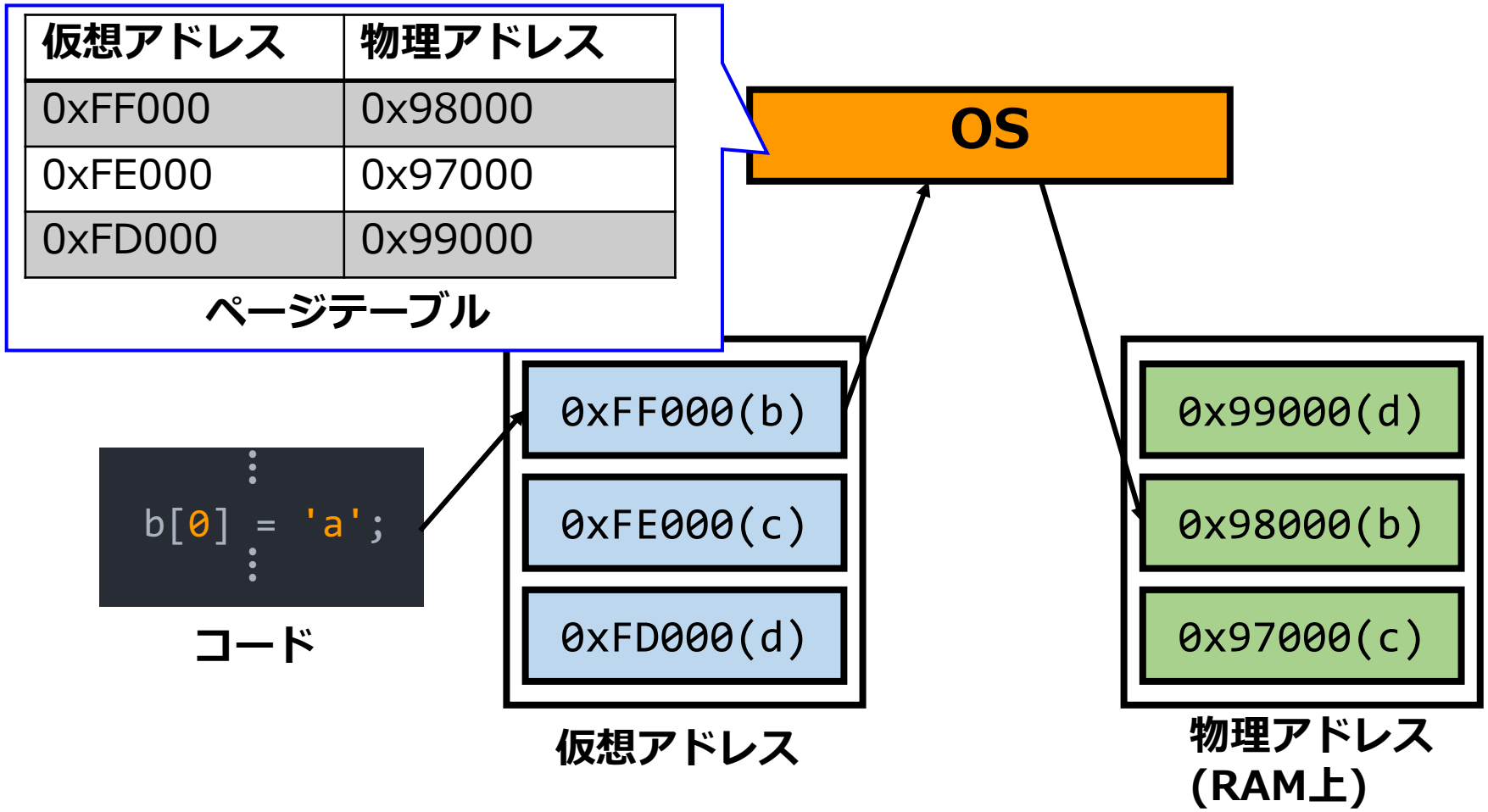
- 仮想記憶を実装するためのアルゴリズム



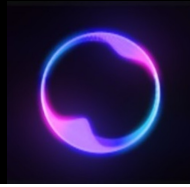
ページテーブル



- OSは**ページテーブル**を用いて**仮想アドレス**に対応する**物理アドレス**を取得する

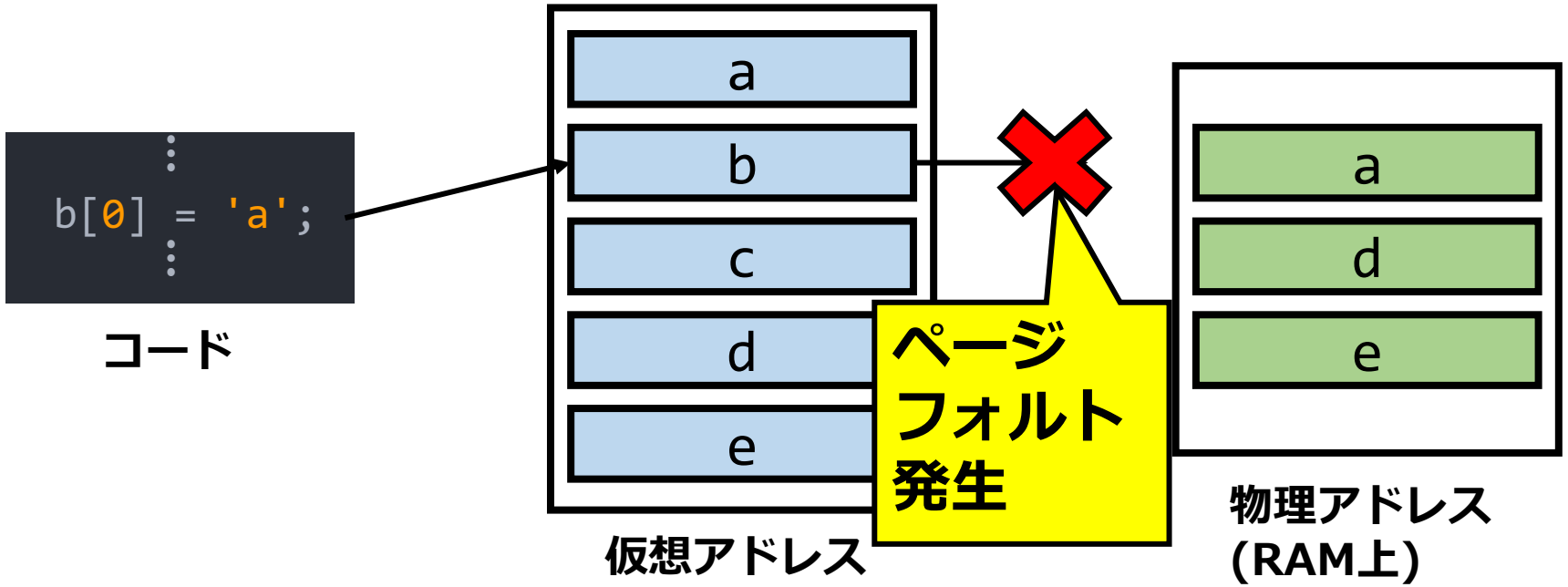


ページフォルト

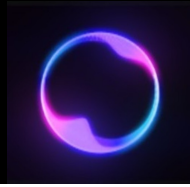


- **ページフォルト**: 以下の状況が発生した場合に発生する、**必ずしも致命的ではない**エラーの事

1. プログラムが**仮想ページ**にアクセスする
2. OSが何らかの原因 (ページアウト、**アクセス制限**等)により**物理アドレス**を**仮想アドレス**から**導出できない**



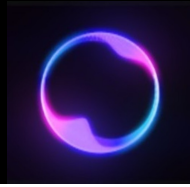
入力依存処理



- ある**処理実行**が**特定の入力データ**によって**条件的に決定される**時、それを**入力依存処理**と呼ぶ
  - 例: ifブロックに囲まれた処理

- 入力依存処理には**2種類**存在する：
  - 入力依存**制御転送**
  - 入力依存**データアクセス**

入力依存制御転送



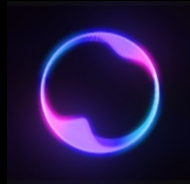
- **変数sによって<u>どちらの関数</u>にアクセスされるかが決定される時、それを「<u>入力依存制御転送</u>」と呼ぶ**

```
char* WelcomeMessage(GENDER s) {
  char *mesg;

  //GENDER is an enum of MALE and FEMALE
  if(s == MALE) {
    mesg = WelcomeMessageForMale();
  }
  else {
    mesg = WelcomeMessageForFemale();
  }

  return mesg;
}
```

入力依存制御転送

入力依存制御転送

入力依存データアクセス



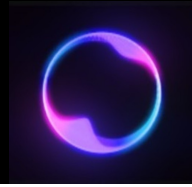
- **変数sによってどちらのデータにアクセスされるかが決定される時、それを「入力依存データアクセス」と呼ぶ**

```
void* CountLogin(GENDER s) {

  if(s == MALE) {
    gMaleCount++;
  }
  else {
    gFemaleCount++;
  }

  return;
}
```

入力依存データアクセス

入力依存データアクセス

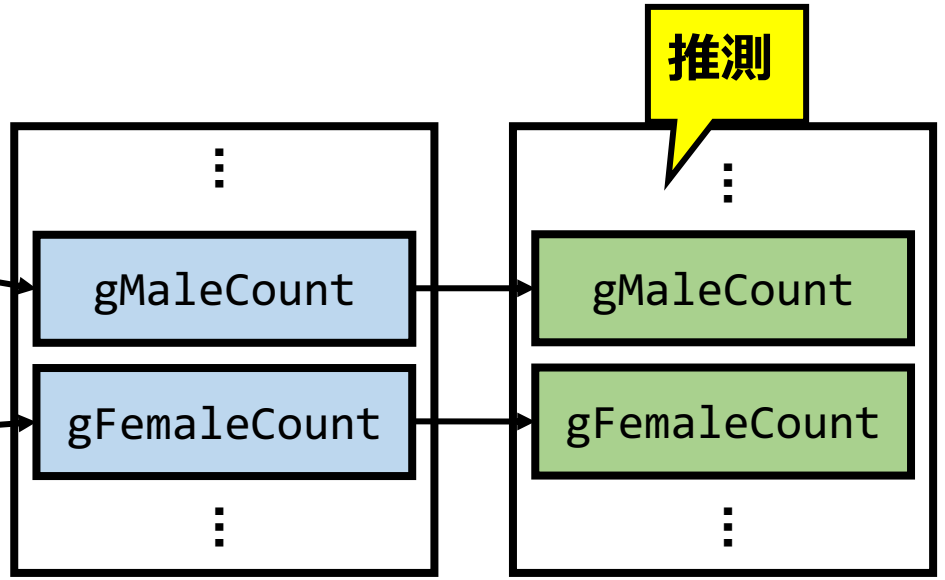
Phase 1. オフライン解析



- **実行バイナリ**や**ソースコード**を解析し、 gMaleCountや gFemaleCountがロードされる**アドレスを推測**する

```
void* CountLogin
(GENDER s) {

  if(s == MALE) {
    gMaleCount++;
  }
  else {
    gFemaleCount++;
  }

  return;
}
```

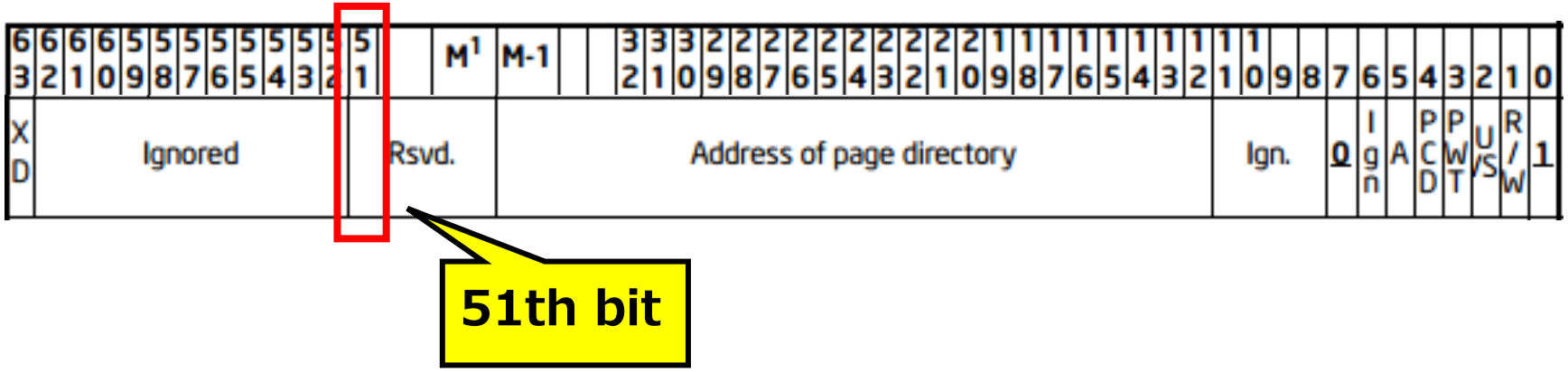
コード



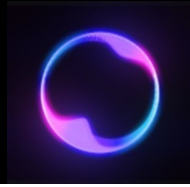
Phase 2. ページフォルト起動 (1/2)

- SGXはページテーブルを**OSと共有**している為、ページテーブルは**容易に改竄できる**

- X86-64のページテーブルエントリの**51ビット目**を立てる事で、**対応するページ**へのアクセスを**全て制限できる**

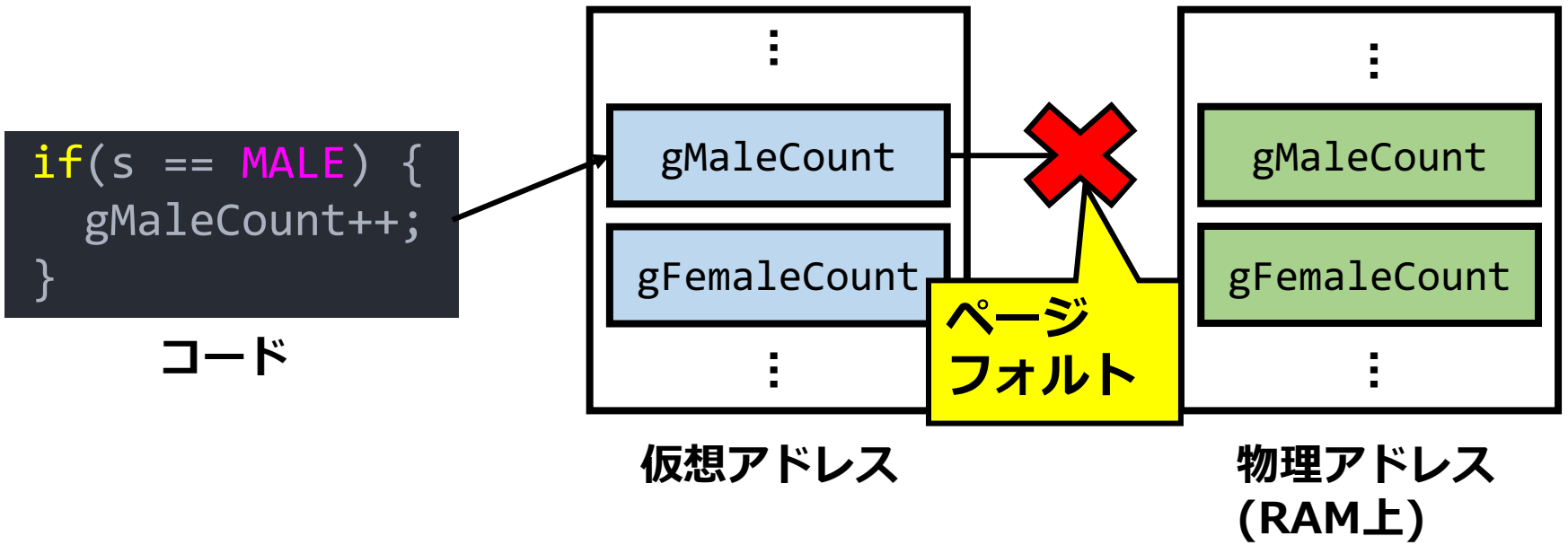


Phase 2. ページフォルト起動 (2/2)

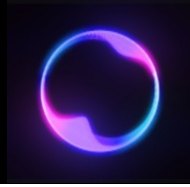


- ページテーブルエントリの**予約ビット**を立てる等により、gMaleCount及びgFemaleCountの載る**ページへのアクセス**を**禁止**する

- **禁止したページ**に載る**データへのアクセス**が発生すると、**ページフォルト**が**発生**する

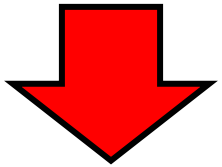


Phase 3. 秘密情報の推定



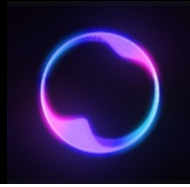
- **ページフォルト**が発生すると、**OS**の**ページフォルトハンドラ**に**ページフォルトアドレス**が伝達される

- 攻撃者は、**攻撃対象の秘密情報**が載っている**アドレス**(あるいは**ページ**)を**オフライン解析**にて取得済みである



ページフォルトアドレスをOSから取得する事により、gMaleCountへのアクセスが発生した事、ひいては“s”がMALEだった事も漏洩してしまう

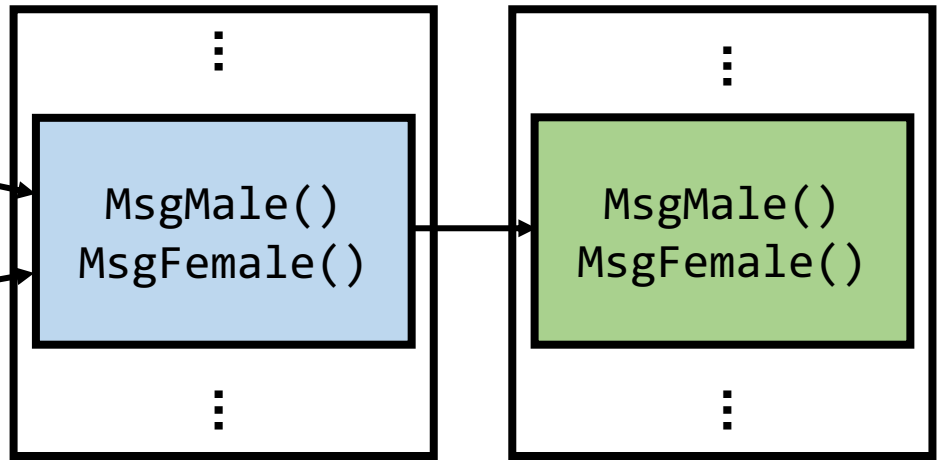
この攻撃を実行する上での困難



- **複数のコードやデータ**が**同一ページ上に載る**可能性が**多分に存在する**ため、その場合単にフォールトの発生したページだけを見るだけではどちらのコードやデータなのか**識別できない**

```
char* WelcomeMessage
 (GENDER s) {
  char *mesg;

  if(s == MALE) {
    mesg = MsgMale();
  }
  else {
    mesg = MsgFemale();
  }

  return mesg;
}
```

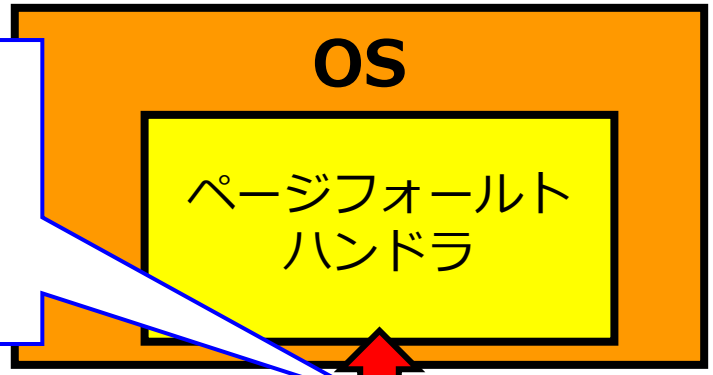
コード



SGXから返されるページフォールトアドレス

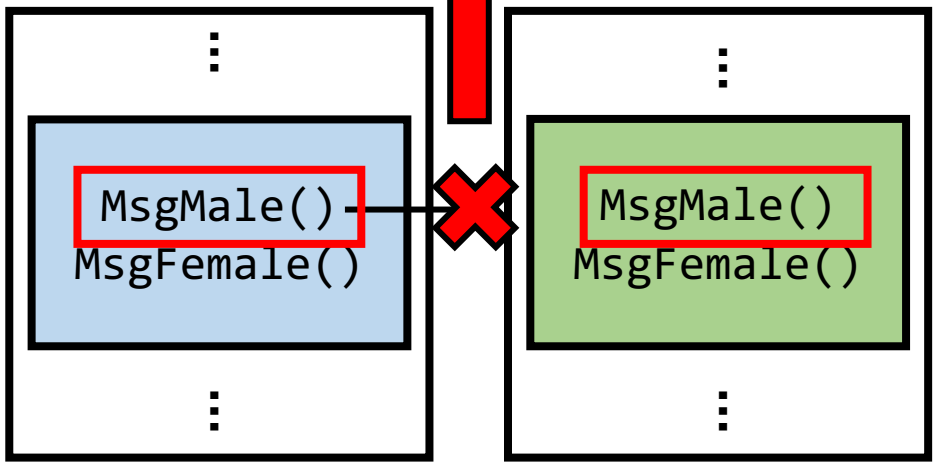
OSが制御可能

通知されるフォールトアドレスは
MsgMale()の完全なアドレスではなく、下位12ビットがゼロ化されている

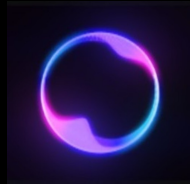


OSはページフォールトの発生したページ番号の粒度でしかアドレスを取得できない

OSは制御不能（Enclave）



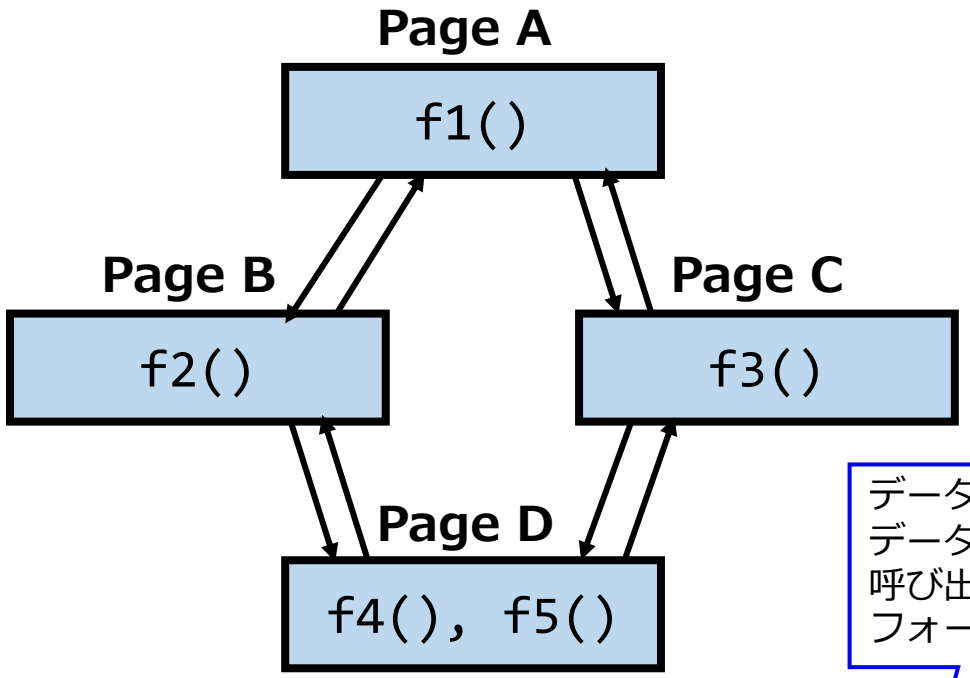
この攻撃を実行する上での困難



- **ページレベルの粒度**でしかページフォールトアドレスからは情報を得られないため、**複数の秘密情報**が**同一ページ上**に**載っている**と、このままでは**識別が出来ない**

- この問題を解決するには、次のページで説明する**ページフォールトシーケンス**という概念を用いる

ページフォールトシーケンス



ページレベルで見た場合の制御転送

データを攻撃対象とする場合は、
データアクセスの直前の関数
呼び出しに対応するコードページ
フォールトシーケンスを頼りにする

A, B, D, B, A, C, D, C, A

f4()　　f5()

コードページフォールトシーケンス

開始点

```
f1 {
   …
   f2();
   …
   f3();
   …
}
```

```
f2 {
   …
   f4();
   …
}
```

```
f3 {
   …
   f5();
   …
}
```

ソースコード

攻撃実践例 – Hunspell（スペルチェッカ）

- Hunspellにより参照される辞書ハッシュテーブルに対して攻撃し、秘密情報である文章を抽出

Folklore, legends, myths and fairy tales have followed childhood through the ages, for every healthy youngster has a wholesome and instinctive love for stories fantastic, marvelous and manifestly unreal. The winged fairies of Grimm and Andersen have brought more happiness to childish hearts than all other human creations.

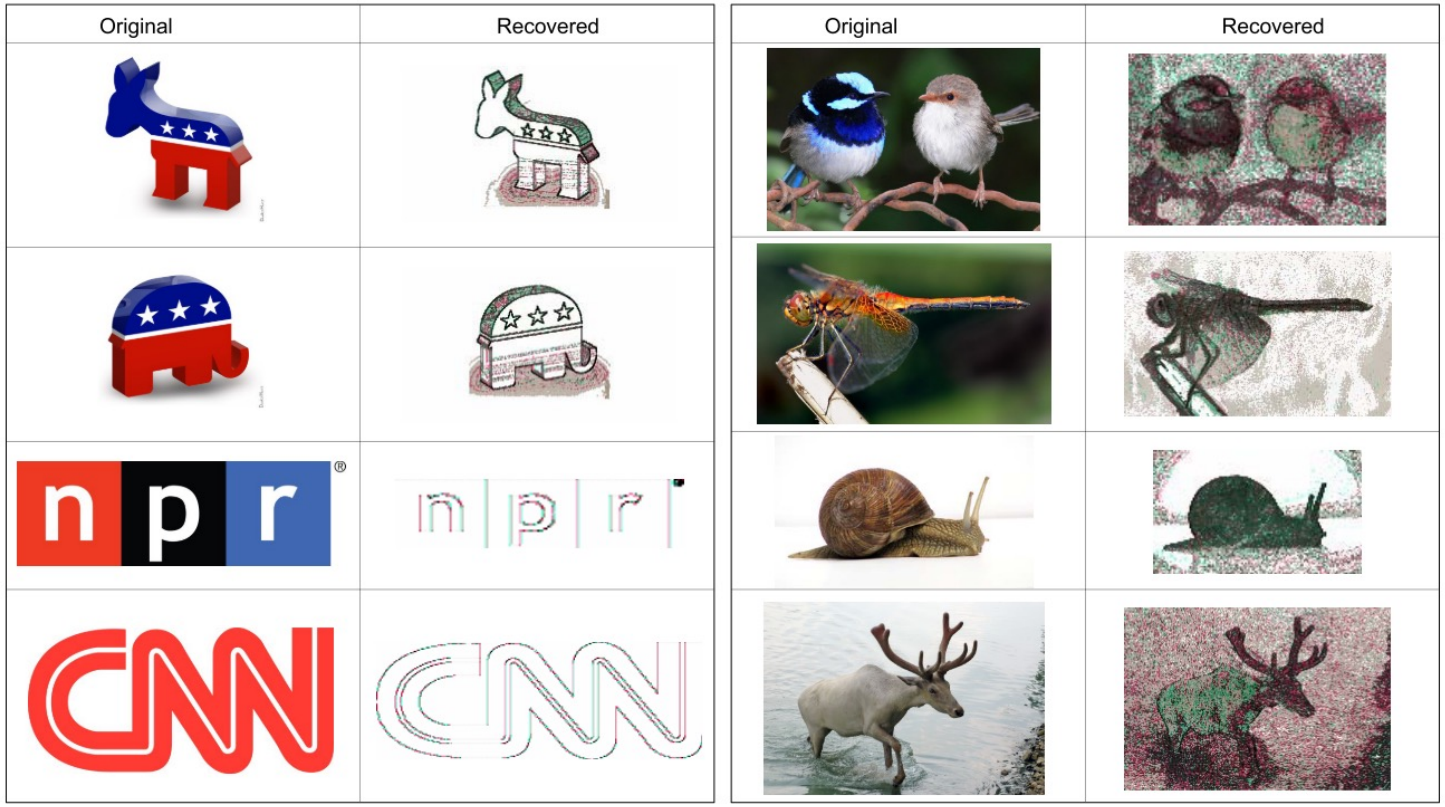
元の文章

folklore *legend* myths and fairy *tale* have *follow* childhood through the *age* for every healthy youngster has a wholesome and instinctive love for [store] fantastic marvelous and *manifest* unreal the [wine] *fairy* of [grill] and Andersen have brought more happiness to childish *heart* than all other human *create*

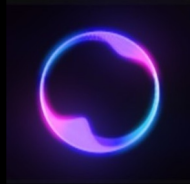
抽出した文章

攻撃実践例 – libjpeg

- JPEGファイルのロード時（デコード時）に呼び出される、逆離散コサイン変換を行う関数に対して攻撃し、秘密情報である画像データを抽出



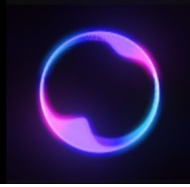
Controlled-Channel攻撃実践（1/6）



■**Controlled-Channel攻撃実践課題**
入力依存データアクセスを行うEnclaveにおいて、分岐先データの載るページのアクセス権を剥奪し、フォールトを発生させて秘密情報の断定を行う実験コードを実装せよ。

但し、OSのページフォールトハンドラを改竄したり中身を見たりするのは難易度が高いため、あくまでも**実験に最適化したEnclave**に対して攻撃を行う。

Controlled-Channel攻撃実践（2/6）



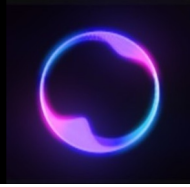
■Controlled-Channel攻撃実践の要件

- 攻撃対象Enclaveは、**uint8_t*型のポインタ2つをグローバル空間**に有している

- また、攻撃対象Enclaveは同じく**グローバル空間**に**1バイト**の**uint8_t型変数**である**秘密情報**を有する

- 攻撃対象Enclaveは、**初期化処理**を行うECALLと、**本処理**を行うECALLの**2つのECALL**を提供している

Controlled-Channel攻撃実践（3/6）

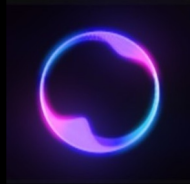


■ **Controlled-Channel攻撃実践の要件**

- 攻撃対象Enclaveの**初期化処理**では、**2つのグローバルポインタ**に対しそれぞれ**ページサイズ**分のバッファを割り当て、バッファを（非ゼロの）**適当な値**（'a'など）で**埋め尽くす**

- また、**sgx_read_rand**関数を用いて**uint8_t型**の**乱数**を生成する。uint8_tの性質上、この値は**0～255**のいずれかになる

Controlled-Channel攻撃実践（4/6）

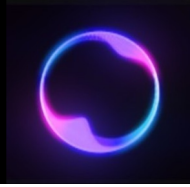


■Controlled-Channel攻撃実践の要件

- **乱数**が**128未満**である場合、**秘密情報**に**0を代入**する。**128以上**である場合は**秘密情報**に**1を代入**する

- 初期化処理ECALL時には、引数として**ページサイズ**を渡しても良い。同時に、Enclave外に**グローバルバッファの仮想アドレス**を**リターン**しても良い

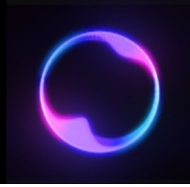
Controlled-Channel攻撃実践（5/6）



■Controlled-Channel攻撃実践の要件

- 攻撃対象Enclaveの**本処理**では、**秘密情報**が**0**であれば**一方のバッファ**に、**1**であれば**もう一方のバッファ**に**アクセス**する（＝**入力依存データアクセス**）

Controlled-Channel攻撃実践（6/6）

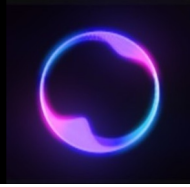


■Controlled-Channel攻撃実践のヒント

- ページアクセスの禁止は、**mprotect関数**をEnclave外で用いる事によって比較的容易に実現できる
  - mprotectに渡すアドレスは、**ページの境界のアドレス**に**一致**していなければならない点に注意

- mprotectでアクセスを禁じたページにアクセスすると**セグフォ**が発生する。この時Enclaveから返ってくるsgx_status_tの値は**SGX_ERROR_ENCLAVE_CRASHED**である
  - つまり、ページフォールトアドレスを見る必要はなく、**クラッシュしたか否か**で判定できるため、**片方の秘密バッファ**の載るページだけ**アクセスを禁じれば十分**

Controlled-Channel攻撃対策の導入実践

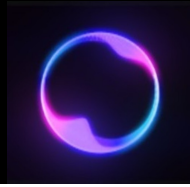


- Controlled-Channel攻撃は、攻撃対象コードに**条件分岐さえ存在しなければ無力**である
  - 条件分岐の排除を含め、タイミング攻撃を含むサイドチャネル攻撃の余地を排する実装方法を**定数時間実装**（Constant-time Implementation）と呼ぶ

- SGX-VaultのEnclave内コードにおいて、どれでも良いので**条件分岐を1つ選び、制御フローが単一**になるように**修正**せよ。

ÆPIC Leak

APIC（1/5）

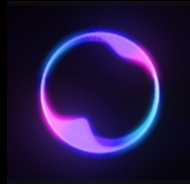


- **APIC**：Advanced Programmable Interrupt Controllerの略で、割り込み処理の高度な制御を行う**割り込みコントローラ**

- **各CPUコア内部**に実装され、**CPUに対し配送**された**割り込み**を処理する**Local APIC**と、**システムに1つのみ存在**し、**外部デバイス**からの**割り込み**を**適切なCPU**に**転送**する**I/O APIC**がある

- SGX攻撃の補助として使われるシングルステップ実行フレームワークである**SGX-Step**も、Local APICのAPICタイマを用いてシングルステップ処理を実現している

APIC（2/5）

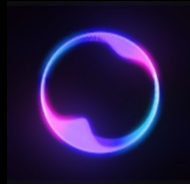


- 最近のAPICは、デフォルトでは**xAPIC**と呼ばれるモードで動作している。xAPICでは、Local APICとのやり取りを行うための**APICレジスタ**を、物理アドレス空間上の4kBの**メモリマップトI/O**（**MMIO**）の形で公開している
  - **MMIO**：メインメモリ上のある特定のアドレスにアクセスすると、対応する入出力機器とのやり取りが出来る仕組み

- より新しいモードとして、**x2APIC**というモードも存在する。割り込み配送の性能向上が図られており、またAPICレジスタには完全に**MSRを通してアクセス**する（**MMIOアクセスの廃止**）

- どちらのモードを用いるかは**OS（root権限）**によって**設定**できる

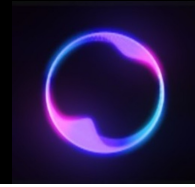
APIC（3/5）



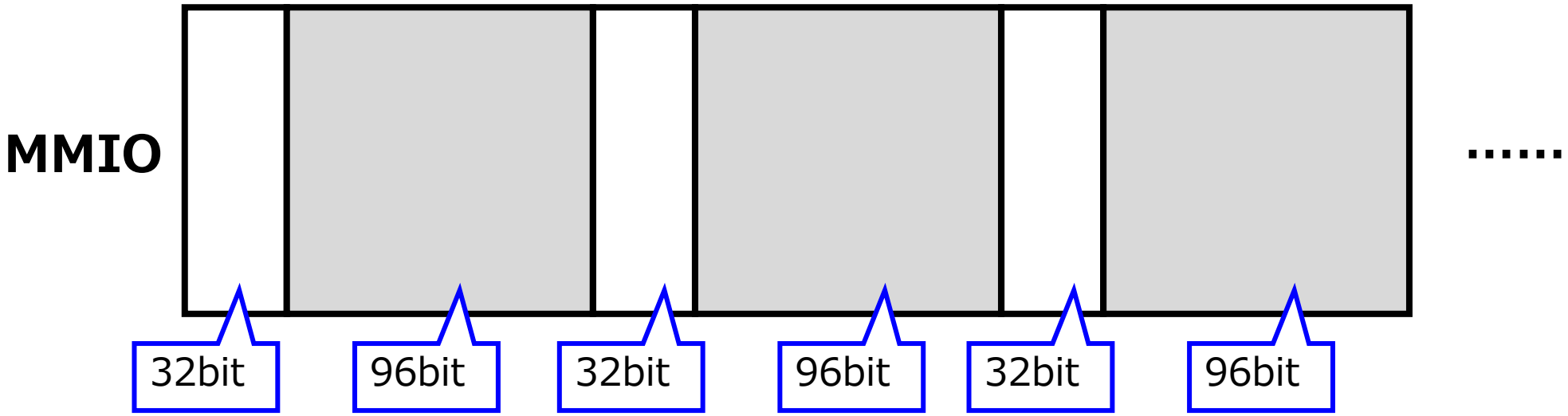
- MMIOのベースアドレスはデフォルトでは物理アドレス0xFEE00000に設定されているが、MSRのIA32_APIC_BASEの値を変更する事でコアごとに対応するMMIOアドレスを変更できる

- APICレジスタは32bit、64bit、または256bitのいずれかの大きさの値を取り扱うが、**xAPICモード**における**メモリマップト領域**では**32bit単位に分割**してマッピングされ、さらに**128bit単位に整列**される

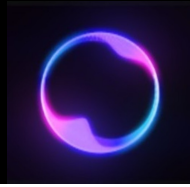
APIC（4/5）



- **32bit**のメモリマップト領域を**128bit**単位に整列するために、**96bit**の**パディング領域**を挟む事によってこれを実現している



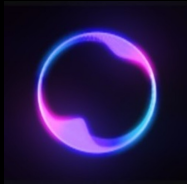
APIC（5/5）



- このパディング領域は文字通り**パディング用**であり、それ以外に何らかの**アーキテクチャ的な定義がなされていない**領域である

- Intelは、この**パディング領域へのアクセス**は**未定義の動作**を引き起こす可能性があるため**行ってはならない**と言及している
  - 普通は、0x00または0xFFの読み出し、システムハング、そしてトリプルフォールト（例外ハンドラの失敗の対応にも失敗した際のフォールト）のいずれかが発生する
  - あくまでもメモリマップト領域である32bitの部分にのみアクセスしAPICとのやり取りを行うのが本来の想定

キャッシュ階層（1/4）



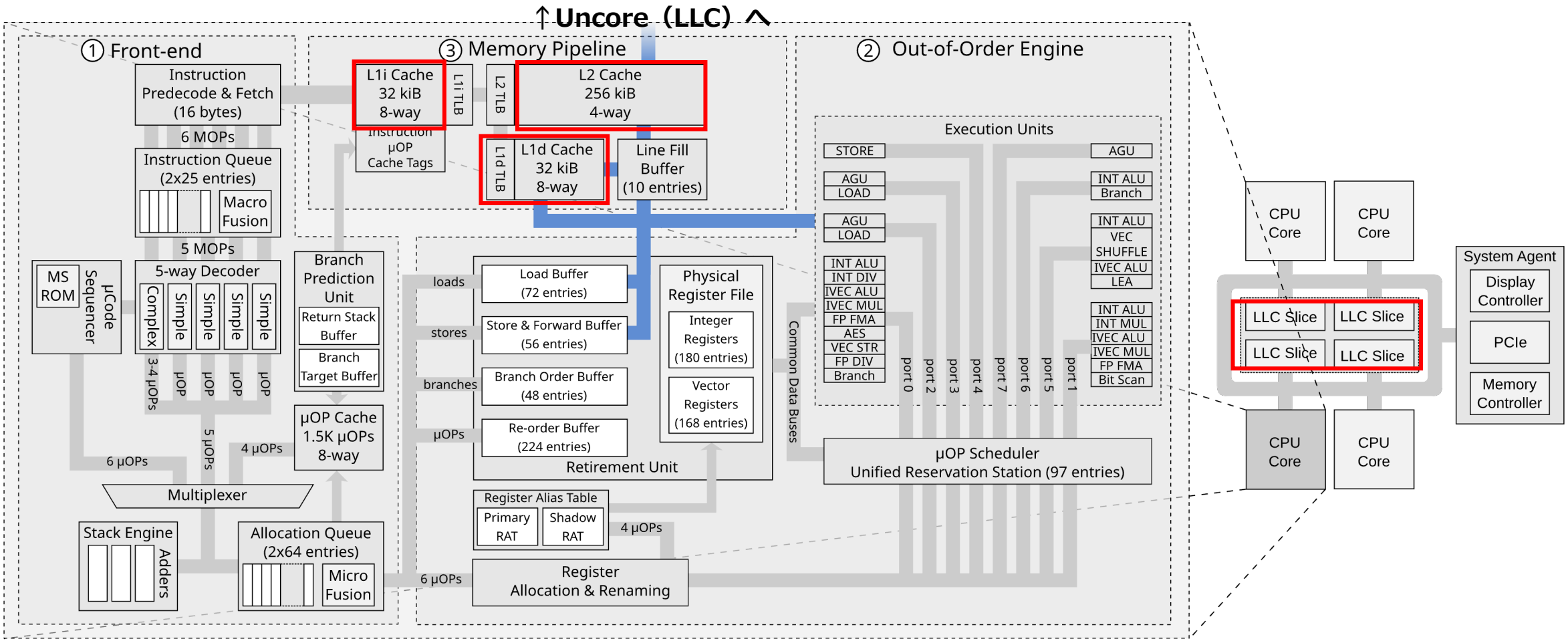
- 頻繁にアクセスするデータや命令を保持する役割を持つ、CPUパッケージ内に存在しメインメモリよりもアクセス時間の速い**キャッシュメモリ**は階層構造になっている

- 最近のCPUでは、CPUの主要部分に近い順に**L1, L2, L3キャッシュ**の3つのキャッシュにより成り立っている

- **L1キャッシュ**には、命令を保持しておく**L1Iキャッシュ**と、**データ**を保持しておく**L1Dキャッシュ**が存在する

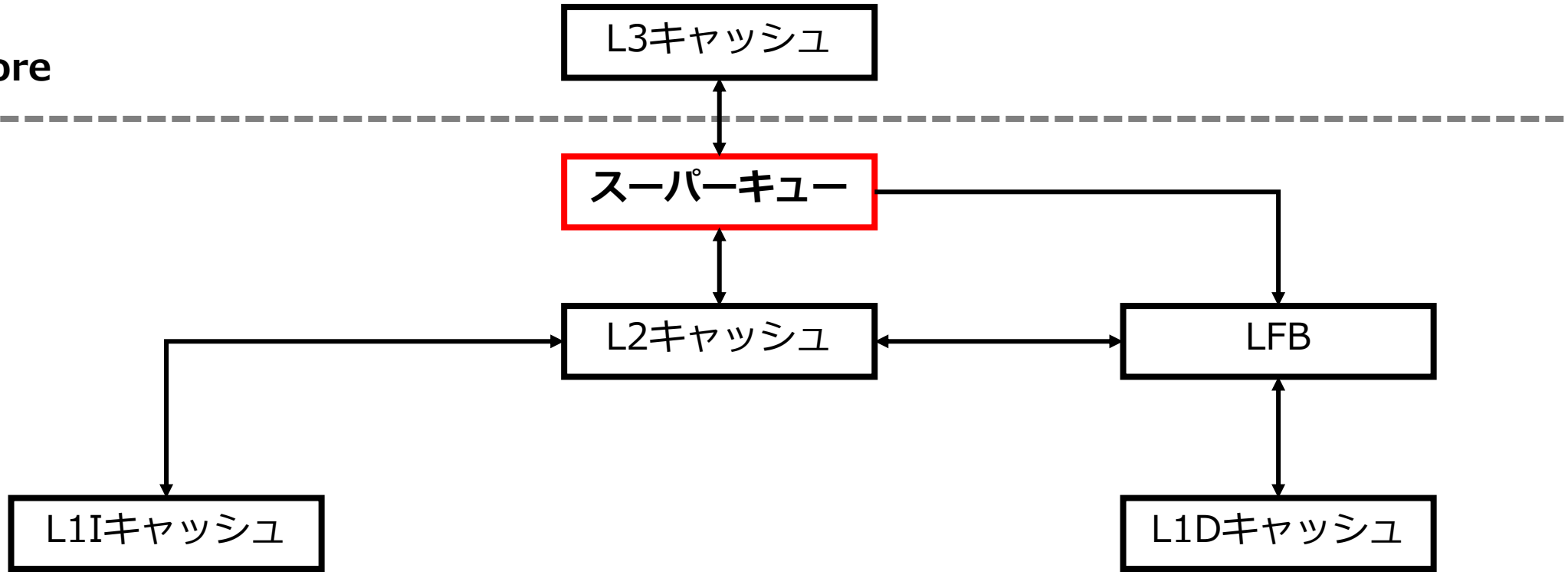
キャッシュ階層（2/4）

- 以下の図はSkylake CPUの構造を示すものである（[4]より引用）

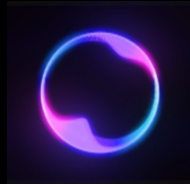


キャッシュ階層（3/4）

- さらに、**L1D-L2間のLFB**に相当するものとして、**L2とL3の間**に**スーパーキュー（Superqueue）**という**架け橋的なバッファ**も存在する



キャッシュ階層（4/4）



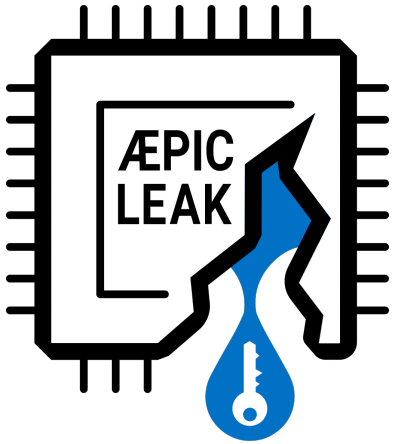
- **APIC**は、**L3キャッシュ**から**L2キャッシュ**に値を持ってくる際に、この**スーパーキューを使用**する

- スーパーキューはキャッシュ間のデータの転送に用いられるものであるため、ユーザ空間、カーネル空間、そして**Enclaveのデータ**等、**あらゆる出処のデータ**が**格納される可能性**がある

ÆPIC Leak（1/4）

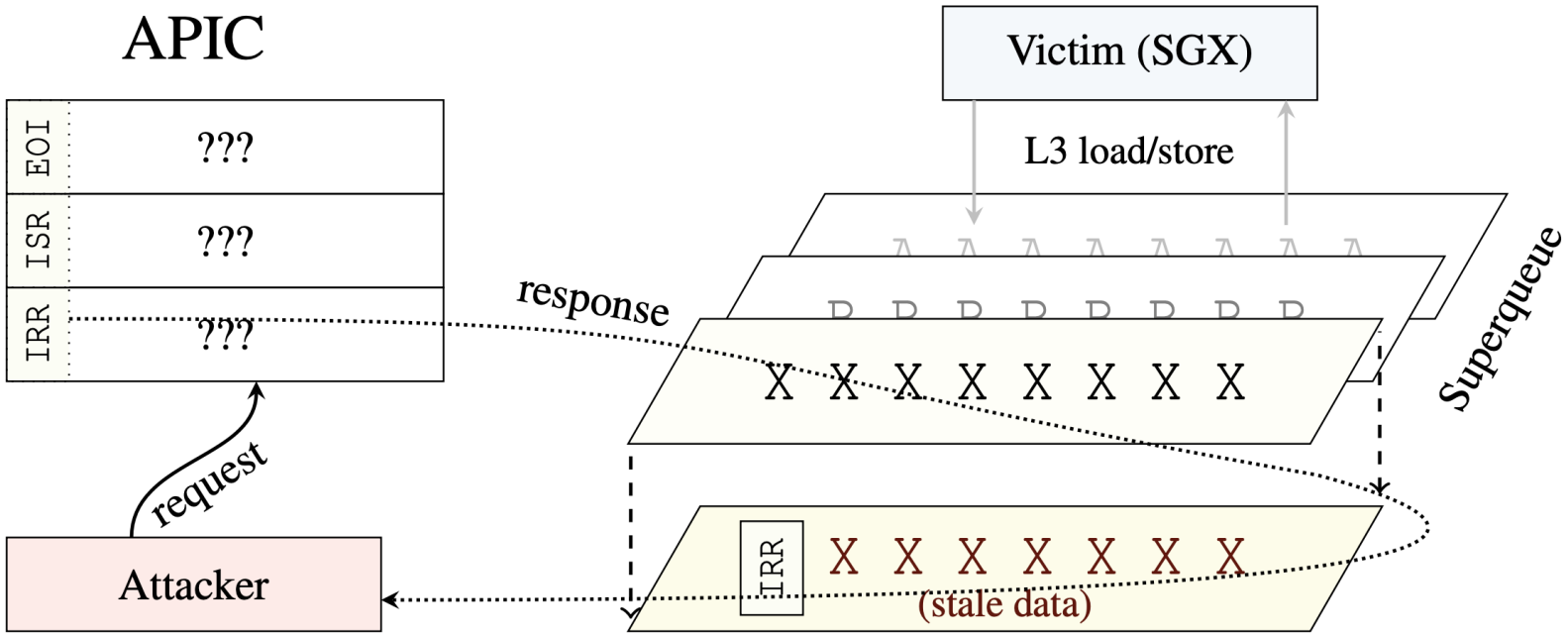
- **Sunny Coveマイクロアーキテクチャ**を搭載するCPUにおいて、**xAPICのMMIOのパディング領域**を読み取ると、**スーパーキューに残留している値**が**漏洩する**バグがある事が判明した
  - Sunny Cove μ-Archを搭載するCPUとしては、第10～13世代Coreシリーズ CPUや、第3世代Xeon-SP CPUが挙げられる

- このバグを悪用し、**Enclave由来**の**スーパーキュー内**の**秘密情報**を**漏洩させる**攻撃が**ÆPIC Leak**である
  - 根本原因は、APICレジスタ（MMIO）の**パディング部分**が**正しく初期化されない**という**アーキテクチャ上のバグ**である
  - 現在発表されているSGX攻撃の中では最新に近い攻撃の1つ



ÆPIC Leak（2/4）

- ÆPIC Leakの概要図（[5]より引用）



ÆPIC Leak（3/4）

- 具体的には、**パディング領域**に**1～4バイト単位**でアクセスを行うと**前述のような漏洩が発生**する
  - 8バイト以上単位でのアクセスでは必ず0xFFが返却される
- APICレジスタは複数存在するが、**アクセスするAPICレジスタ**と**その内容**に**相関は存在しない**
- 一方、APICアドレスに対応する**キャッシュライン上のオフセット**は、**漏洩させるデータ**の**キャッシュライン上のオフセット**に**一致**する
  - **キャッシュライン**：キャッシュの構成単位（通常**64バイト**）
  - 例えばAPICベースアドレスから0x10のオフセットの場所にアクセスすると、攻撃対象キャッシュラインの0x10のオフセットの値が漏洩するイメージ

ÆPIC Leak（4/4）

- 128bitの内パディング部分でない**32bitのメモリマップト領域**は**正常に初期化**されるため、**64バイトのキャッシュライン**であれば**後半48バイト**（後半**3/4**）のみが**漏洩**する

- さらに、**128の倍数のアドレス**から始まる**キャッシュライン**からのみ**漏洩**させられる
  - キャッシュラインは**通常2つペアで転送**されるため、**2つ目**のキャッシュライン（アドレス：64+128×n）がまず**転送**され、その後**1つ目**（アドレス：128×n）で**上書きする形になる**からであると考えられる

- 結果的に、ÆPIC Leakでは$\frac{96}{128} \times \frac{1}{2} = 0.375$、即ち**任意のページ**の**37.5%**を漏洩させる事が出来る

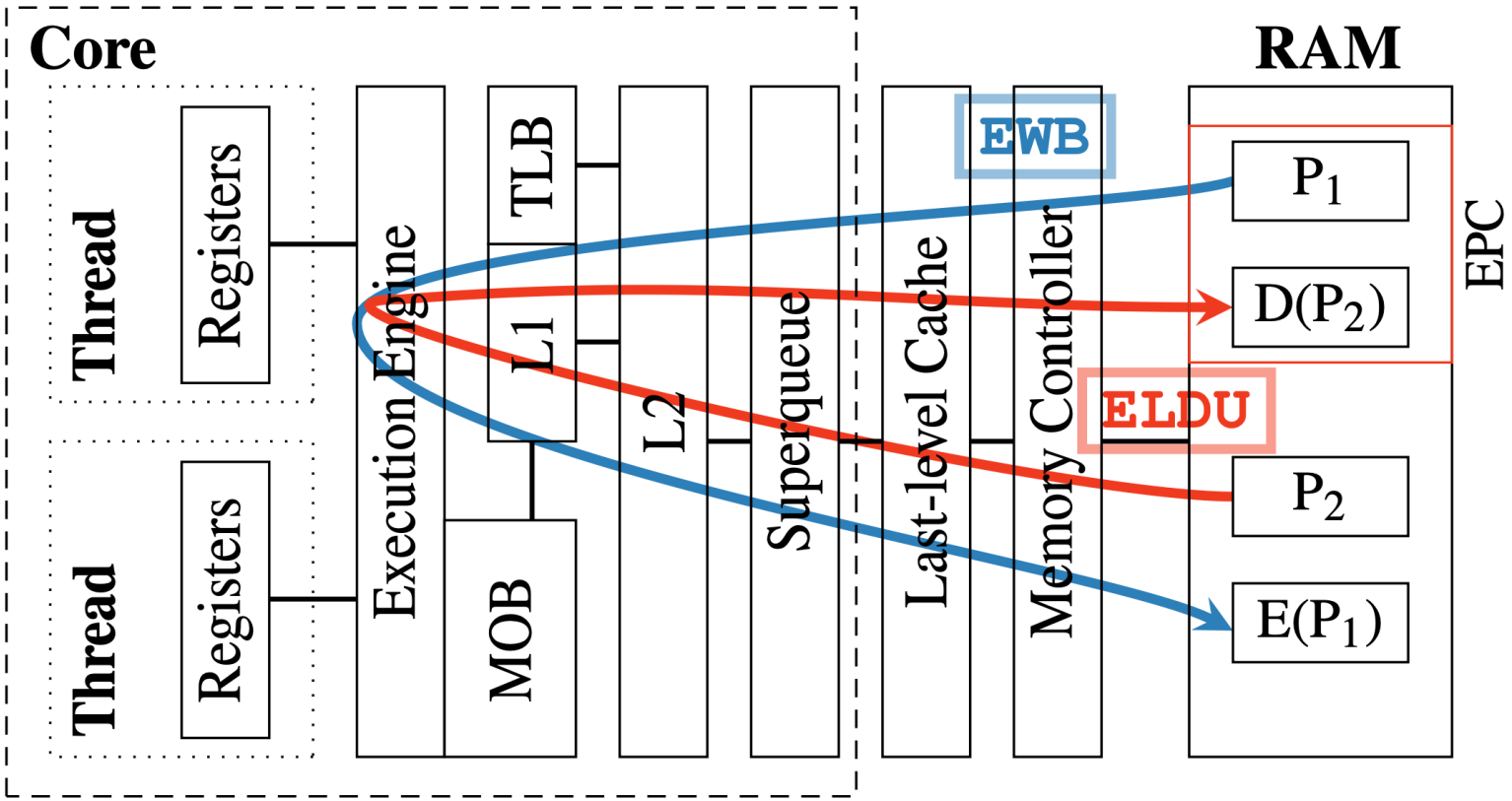
ÆPIC Leakの補助手法（1/3）

- Foreshadow攻撃（後述）でも使用されている手法であるが、**秘密情報が載るEPCページ**を**EWB**と**ELDU**で**ページ退避&ロード**を繰り返す事で、強制的に秘密情報の載るページの内容を**キャッシュ階層**（**スーパーキュー含む**）に**持ち込む**事が出来る

- Foreshadow攻撃の論文ではこの手法に対する名前をつけていないが、ÆPIC Leakの論文で**Enclave Shaking**と名付けられている

ÆPIC Leakの補助手法（2/3）

- Enclave Shakingの概要図（[5]より引用）。Enclave Shakingにより、秘密情報の載るページの内容がキャッシュ階層に浸透する様子を示している



ÆPIC Leakの補助手法（3/3）

- 別の補助手法として、ある**攻撃対象**の載るものと**並行して実行**されている、**同一物理コア内**の**論理コア**（=**ハイパースレッド**）、即ち**兄弟スレッド**（Sibling Thread）を**悪用**する手法がある

- **攻撃対象**に**並行して実行**される**兄弟スレッド**にて、攻撃対象と**無関係なページ**の**ページオフセットx**に**連続的にアクセス**すると、**攻撃対象ページのオフセットxの値**が**漏洩**する**可能性を上げられる**

- 論文では、この手法を**Cache Line Freezing**と呼んでいる

ÆPIC Leak攻撃例 – SECRET Networkへの攻撃（1/7）

- ÆPIC Leakの攻撃例として、SGX Failのセクションでも取り上げた**SECRET Network**の**コンセンサスシードの抽出**を行う攻撃を取り上げる
  - 当該セクションで説明した通り、SGX Failの論文で実験として実際に行われた攻撃である

- 早い話、コンセンサスシードを暗号化しているSGX_FILEのAES鍵（**IPFSL鍵**）に対して**ÆPIC Leakを仕掛け**、IPFSL鍵を抽出し**コンセンサスシードを復号**する事を考える

（復習）SECRETにおけるコンセンサスシード（1/4）

- SECRETでは、前述の様々な保護機能で使用する**鍵**は、全て親玉（**マスター秘密鍵**）である**コンセンサスシード**から導出される

- スマートコントラクトへのメッセージ送信時は、ユーザは**コンセンサスシードから導出**した**公開鍵**を用いて**暗号文**を**トランザクション**に含める
  - 一方、**秘密鍵**は**MRSIGNERポリシのシーリングデータ**として、チェーン全体に複製（デプロイ）される

- また、口座残高等の現在の状態を暗号化する鍵もまたコンセンサスシードから導出される

（復習）SECRETにおけるコンセンサスシード（2/4）

- 当然、この**コンセンサスシード**が**万が一にも漏洩**すると、SECRETが売りにしている**あらゆる保護機能が無力化**される

- さらに、このコンセンサスシードは原則として**永続的かつ不変**のものであるため、もし漏洩すると**極めて面倒な事になる**
  - **ブロックチェーンそのもの**を**分裂**させなければならない**ハードフォーク**でのみ対応する事が出来る

（復習）SECRETにおけるコンセンサスシード（3/4）

- コンセンサスシードは、独自に用意した鍵により**Intel Protected File System Library**（以下、IPFSL）を用いて128bit AES/GCMで暗号化され**Enclave外にストア**される

- **IPFSL**：**SGX_FILE**型という**Enclave内**で**直接扱えるような**FILE構造体に基づき、**暗号化した状態でのファイル入出力**（sgx_fopen等）を提供する機能
  - Enclave内から**直接Enclave外に保存**できる、**暗号化鍵を指定できる**等の部分でシーリングと異なる
  - 本ゼミでは使用していないが、includeとリンクさえ行えば普通に**デフォルトのSGXSDK**で**利用可能**

（復習）SECRETにおけるコンセンサスシード（4/4）

- IPFSLでコンセンサスシードを暗号化する際に使用した鍵は**シーリングでストア**される
  - 鍵自体はAESで使う普通の128bitの共通鍵

- SGX Failでは、この**IPFSL用の鍵**がEnclave内で**コンセンサスシードの暗号化及び復号**に**使用されている最中**に**攻撃を仕掛け**、IPFSL用鍵を抽出する手法を選択した
  - **IPFSL鍵が漏れる**と**簡単にコンセンサスシードが復号**出来てしまい、**SECRET上のあらゆる秘密情報**が**究極的には暴かれてしまう**

ÆPIC Leak攻撃例 – SECRET Networkへの攻撃（2/7）

- まず、Controlled-Channel攻撃の**ページフォールトシーケンス**手法を利用し、IPFSLがAES鍵により**コンセンサスシードを復号**する関数の所まで**実行を進める**
  - 厳密には、復号にあたり鍵伸長処理を行うk0_aes_DecKeyExpansion_NI()関数の実行まで進める

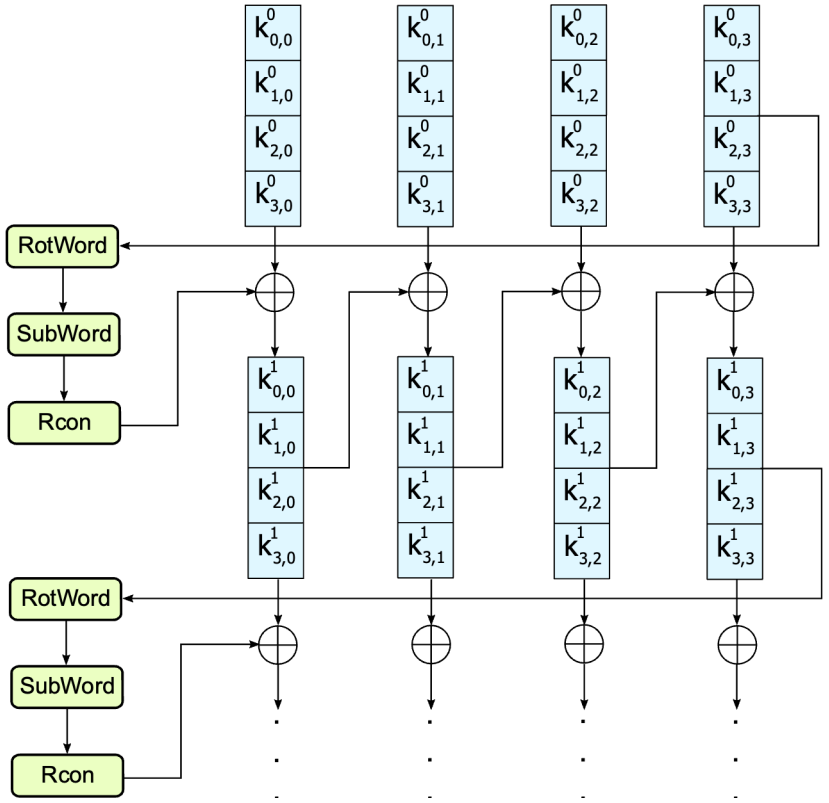
- この状態で**Enclaveを中断**し、**ÆPIC Leak攻撃**を**実行**する

ÆPIC Leak攻撃例 – SECRET Networkへの攻撃（3/7）

- ÆPIC Leakを仕掛けると、**N**ラウンド目と**N+1**ラウンド目のラウンド鍵の、それぞれ**後半3ワード**が**復元**できる
  - 1ワード：サイズは4バイトで、AESブロックである4×4行列の1行または1列に相当する。ワード換算では128bitは4ワードになる
  - 先頭1ワードが抽出できないのは、前述の37.5%のリーク率が原因である

ÆPIC Leak攻撃例 – SECRET Networkへの攻撃（4/7）

- ここで、AESの鍵伸長では、**N+1**ラウンド鍵の**第2ワード**を導出するために、**N+1**ラウンド鍵の**先頭ワード**と**N**ラウンド鍵の**第2ワード**とでXORを取っている（図は[6]より引用）



ÆPIC Leak攻撃例 – SECRET Networkへの攻撃（5/7）

- ここで、AESの鍵伸長では、**N+1**ラウンド鍵の**第2ワード**を導出するために、**N+1**ラウンド鍵の**先頭ワード**と**N**ラウンド鍵の**第2ワード**とでXORを取っている

- つまり、nラウンド鍵の第iワードをW_i^nと表現すると、上記は

$$W_2^{N+1} = W_1^{N+1} \oplus W_2^N$$

となり、XORの対称性により

$$W_1^{N+1} = W_2^N \oplus W_2^{N+1}$$

が成立する

ÆPIC Leak攻撃例 – SECRET Networkへの攻撃（6/7）

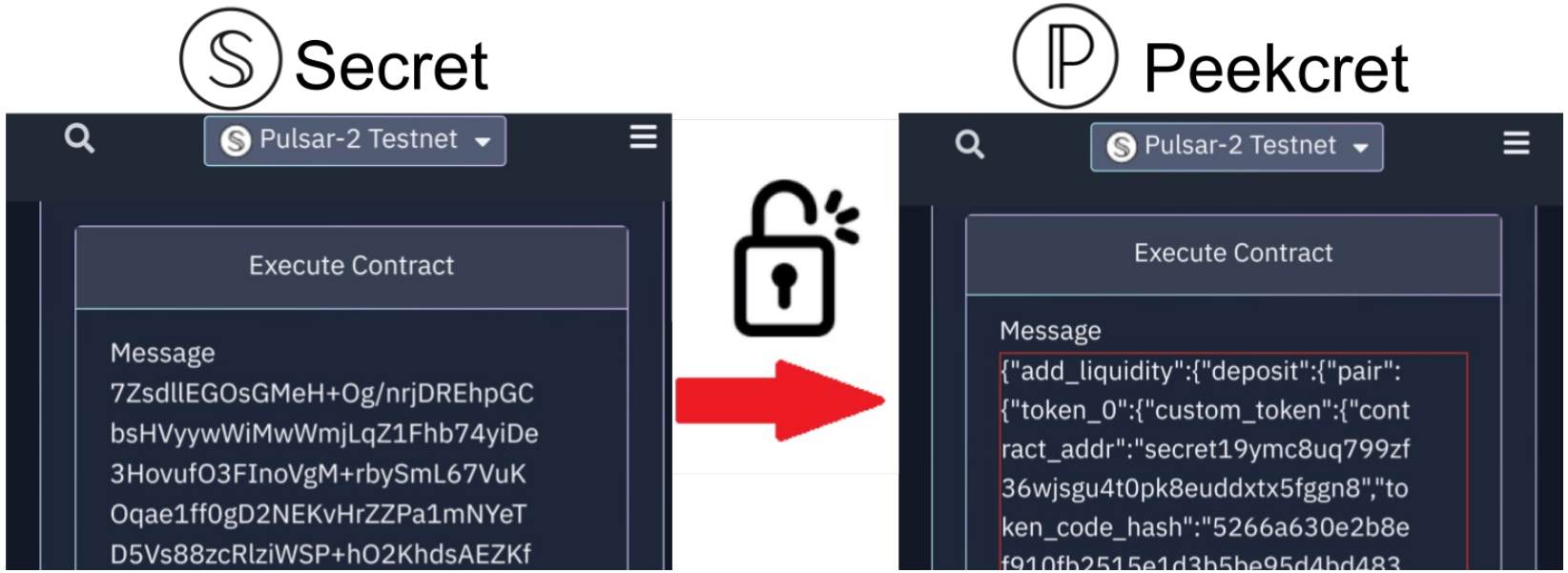
- これは即ち、**N**ラウンド鍵と**N+1**ラウンド鍵のそれぞれ**第2ワード同士をXOR**すれば、**N+1**ラウンド目の**先頭ワードが復元**でき、**N+1ラウンド鍵**が**完全に抽出**できる事になる

- 使用する**AES関数の仕様**（S-BoxやRcon等）は**公開情報**であるため、あるラウンド鍵から**1つ前のラウンド鍵**を**導出する材料**はこれで**全て揃う**事になる

- どのラウンド鍵を抽出したかは分からないため、128bit AESのラウンド数である**10通りのブルートフォース**を行い、最終的に**IPFSLのAES鍵を復元**し**コンセンサスシードを復号**できてしまう

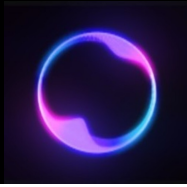
ÆPIC Leak攻撃例 – SECRET Networkへの攻撃（7/7）

- 解読したコンセンサスシードを使用し、以下のように**SECRETの**
  ブロックのトランザクションの復号に**成功**している
  （図は[7]より引用）



Foreshadow（L1 Terminal Fault)

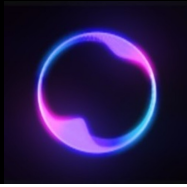
Foreshadow攻撃（1/4）



- **Enclave内のデータ**は、通常のメインメモリの値同様**キャッシュ**に**ストア**される

- ところで、通常**Enclave内**のメモリに対して**外からアクセス**すると、**読み出し値**は**0xff**となり、**書き込み**は**無効化**される
  - この仕様を**アボートページセマンティクス**（以下、**APS**）という

Foreshadow攻撃（2/4）

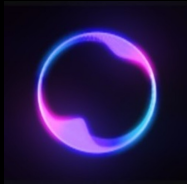


- しかし、APSが適用されるような命令でそれ以上アドレス解決の行われないページフォールト（=**ターミナルフォールト**）が発生すると、APSの適用前に**投機的実行**が発生する

- **フォールトの発生した命令**でアクセスした**アドレスに対応する値**がL1Dにキャッシュされていると、この**投機的実行**においてその値が**過渡的に使用**されてしまう

- 命令リタイアに伴う**投機的実行結果の棄却の前**に**この情報依存の痕跡**を**キャッシュに残す**事で、**過渡的な値**を**キャッシュサイドチャネル的に抽出**出来てしまう

Foreshadow攻撃（3/4）



- この仕様を悪用し、予め**Enclaveの秘密情報**を**L1D**に**キャッシュ**させておき、そのアドレスにアクセスするEnclave外の命令で**ターミナルフォールト**を起こす事を考える

- すると後続の過渡的（投機的）実行で**秘密依存の痕跡**が**キャッシュに残る**ため、結果的に**秘密情報を抽出**できてしまう

Foreshadow攻撃（4/4）

- この手段によりEnclave内の秘密情報を抽出する攻撃こそが**Foreshadow攻撃**である
  - **Meltdown型攻撃**の一種に分類される
- **L1D**に存在するデータが**ターミナルフォールト**に伴い漏洩する為、Intel公式では**L1 Terminal Fault**（L1TF）と呼ばれている
- 頑張れば**非root**でも**攻撃を成立**させられる

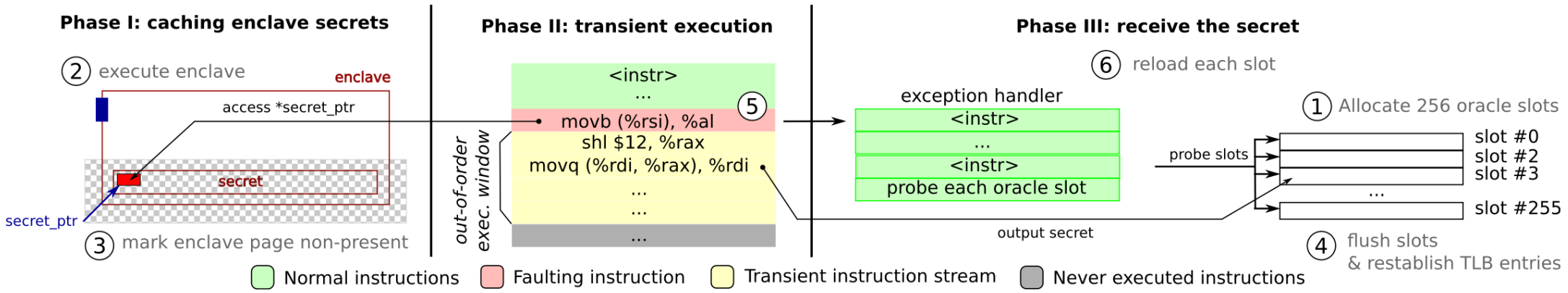
Foreshadow攻撃のごく簡単な攻撃例（1/2）

- ここではまず、Foreshadowの概念実証的な攻撃コードについて説明を行う

- Foreshadowは、主に以下の3フェーズにより構成される：
  - **フェーズI**：Enclave秘密情報のL1Dへのキャッシュ
  - **フェーズII**：ターミナルフォールトの誘発と過渡的実行
  - **フェーズIII**：秘密情報の抽出

Foreshadow攻撃のごく簡単な攻撃例（2/2）

- Foreshadow攻撃のフロー概要図（[11]より引用）



Foreshadow - 事前準備（1/2）

- 秘密情報の抽出を行うための**オラクルバッファ**と呼ばれる**監視用配列**を用意する
  - 秘密情報の抽出は**1バイト**（0x00～0xFFの**256通り**ある）**ずつ行う**

- オラクルバッファは、**1バイトずつ抽出**するために用いるため、1バイトが取り得る値の数（256通り）に合わせて**256スロット**を有している

- **1スロット**あたり**4096バイト**（=**ページサイズ**）であるため、事実上のオラクルバッファのサイズは256×4096バイトである

Foreshadow - 事前準備（2/2）

- フェーズIIの過渡的実行により、**オラクルバッファ**の「**秘密バイト×4096**」の**インデックス**にアクセスさせ、その**インデックスに格納されている値**を**キャッシュ**に残させる

- **1スロット**あたりの**サイズ**が**ページサイズ**である理由は、キャッシュラインプリフェッチャによる誤作動で**正しくないスロット**に**対応**する**痕跡が残る**のを防止するためである

Foreshadow - フェーズI

- フェーズIでは、**Enclave内**に存在する**秘密情報**を**L1Dにキャッシュ**させる
  - FLUSH+RELOADを仕掛ける（最初はキャッシュが皆無でなければならない）ため、キャッシュさせる前に予めオラクルスロットの**キャッシュをフラッシュ**しておく（clflush命令でフラッシュ可能）

- 秘密情報の載るEPCページに対しENCLS命令であるEWBとELDUを繰り返す事で、攻撃対象の処理に関係なく、無理矢理L1Dに秘密情報をキャッシュさせる事が出来る
  - 後述の**ÆPIC Leak**攻撃でも使用される、**Enclave Shaking**と呼ばれる手法

Foreshadow - フェーズII（1/3）

- フェーズIIでは、**ターミナルフォールト**を発生させるために
  アクセスする（秘密情報が載る）**Enclaveページ**への**アクセス権を剥奪**する

- Controlled-Channel攻撃の時と同様、これは**mprotect**関数で
  簡単に実現できる：

```
//PTEのPresentビットをクリアしアクセス不能にする
mprotect( secret_ptr &~0xfff, 0x1000, PROT_NONE );
```

Foreshadow - フェーズII（2/3）

- 以下のコードを用いてForeshadowによる秘密情報の抽出を行う事を考える：

```
void foreshadow(uint8_t *oracle, uint8_t *secret_ptr)
{
    uint8_t v = *secret_ptr;
    v = v * 0x1000;
    uint64_t o = oracle[v];
}
```

- この攻撃コードは、**攻撃者**が**自前で用意**し**攻撃対象のマシン**で**実行**させられるため、Foreshadowは**攻撃対象のコードに依存しない**というメリットが存在する

Foreshadow - フェーズII（3/3）

- mprotectで**アクセス権を剥奪**した事により、3行目の uint8_t v = *secret_ptr;で**ターミナルフォールト**が発生する

- ここで、**ターミナルフォールト**に伴う**過渡的実行**において、4・5行目の**vの値**として、**L1D**から持ってきた**秘密バイト**が**過渡的に使用**されてしまう
  - フェーズIでL1Dにこの秘密バイトをキャッシュしていないと、この過渡的なL1Dからの取得が発生しなくなる

Foreshadow - フェーズIII

- 5行目でオラクルバッファの**秘密バイト×4096**のインデックスにアクセスしているため、このインデックスのアドレスと格納されている値が**キャッシュ**される

- 命令リタイアにより**過渡的実行の値は棄却**されるが、既に**キャッシュに秘密依存の値**が**残っている**ので、**FLUSH+RELOAD**キャッシュサイドチャネル攻撃により**秘密バイトが抽出**できる
  - オラクルバッファの**全スロットにアクセス**し、**アクセス時間**が**短かかったスロット**の**インデックスが秘密バイト**である

Foreshadow攻撃例 – LEへの攻撃（1/10）

- 既にLEは**Deprecated**で**ほぼ形骸化している**が、論文での説明が一番充実しているのがLEへの攻撃であるため、LEに対する攻撃について説明する
  - ここでのLEはref-LEではなく、Intel公式により署名されている、昔ながらのLEである

- ENCLS命令の**EINIT**により**Enclaveを初期化**する前には、**LE**から**EINITTOKEN**構造体を受け取らなければならない
  - EINITTOKENには、起動しようとしているEnclaveのMRENCLAVEやMRSIGNER等が同梱されている

Foreshadow攻撃例 – LEへの攻撃（2/10）

- LEは、以下の条件を満たすEnclaveに対し起動許可を与える実装になっている
  - 対象Enclaveが**デバッグモード**であるか、対象Enclaveの**MRSIGNER**が**Intel**によって**ホワイトリスト**（Intelのプライベート署名鍵で署名される）**に登録**されているかのどちらかである
  - 対象Enclaveが**PKへのアクセス権**のような、**特権的**で**Intelのみが使用可能な権限**を持っていない

- 起動を許可すると判断したら、LEはEINITTOKENの一部（MRENCLAVE、MRSIGNER等）に対する**128bit AES/CMAC**を、**起動キー**（Launch Key）により算出しEINITTOKENに添付する
  - 起動キーは、**LAUNCHKEY属性**の付与されているEnclave（=**LE**）のみが**直接アクセス**する事が出来る

Foreshadow攻撃例 – LEへの攻撃（3/10）

- LE自体のMRSIGNER値はプロセッサにハードコーディングされているため、**LE自体**は**起動許可処理**を**必要としない**
  - ref-LEでは、MSRのIA32_SGXPUBKEYHASH0～3にそのMRSIGNER値を書き込む事で同様に起動許可のスキップを行える

- LEからEINITTOKENを受け取ったら、**EINIT**は内部で**起動キーを導出**し、付与されたMACを検証する
  - MACの検証に失敗した場合は起動はその時点で中止される
  - そもそも起動許可が与えられない場合、署名（MAC）は付与されない[12]

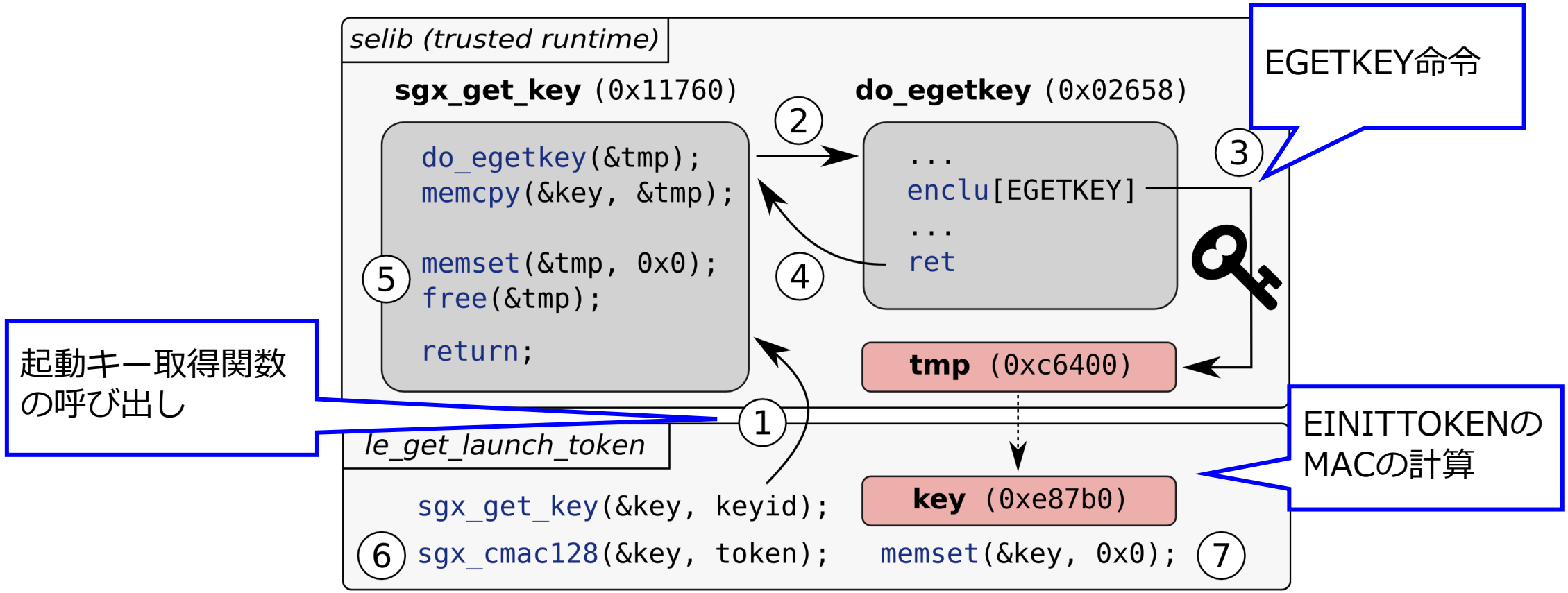
Foreshadow攻撃例 – LEへの攻撃（4/10）

- もしForeshadow攻撃により**起動キーを漏洩**できた場合、完全にLEを迂回して**EINITTOKENを偽造**する事が出来てしまう
  - 本来**起動許可されていないEnclave**を**起動**したり、**PKへのアクセス権**を**不正に付与**してEnclaveを起動させたり出来てしまう

- ただし、レポートキーの導出時と似たような話として、乱数要素を鍵に与える**KeyID**が**EGETKEYごとに変わる**ため、**1回**のLEの処理で**鍵全体を抽出**する必要がある
  - よって、従来型のSGXに対するサイドチャネル攻撃のように、同じ処理を何度も繰り返して鍵全体を復元するというアプローチは取れない

Foreshadow攻撃例 – LEへの攻撃（5/10）

- 以下の図は、対象Enclaveの同一性情報等を渡し、起動キーによるMACが付与されたEINITTOKENを取得するためのLEの関数の動作概要を示したものである（[11]より引用）：



Foreshadow攻撃例 – LEへの攻撃（6/10）

- 前ページの図の内、起動キーを格納するバッファは**tmp**バッファと**key**バッファの2つが存在する

- 論文では、Foreshadowの攻撃性能を実証するために、より短命な**tmpバッファ**から**起動キーを漏洩**させるケースを考えている

- 攻撃を行うにあたり、Controlled-Channel攻撃で登場した**ページフォールトシーケンス**に基づく判断や、**SGX-Step**という（マシン語レベルでの）**シングルステップ実行フレームワーク**を活用している

Foreshadow攻撃例 – LEへの攻撃（7/10）

- オフライン（事前調査）フェーズでは、LEをSGX-Stepで**シングルステップ**し、同一命令における**AEXを連発**させて**SSA**から**CPU内部の状態全体**を**ダンプ**する（詳細は割愛）

- この攻撃手法を**ゼロステップ処理**と呼び、Foreshadow以外の様々なSGX攻撃においても頻繁に使用される

- このオフラインフェーズにより、攻撃対象である**tmpのアドレス**及び**関心のあるコードの位置**を**決定的に把握**できる

Foreshadow攻撃例 – LEへの攻撃（8/10）

- **オンライン**（攻撃本番）フェーズでは、まず前述の図の③と④の間（EGETKEY発行とdo_egetkeyからのリターンの間）でEnclaveに**割り込み**を加える

- sgx_get_keyとdo_egetkeyで**交互**に**コードページ**の**アクセス権を剥奪**するプログラムを実行する事で、ページフォールト回数からdo_egetkey命令の**リターン位置**を**確実に特定**できる
  - オフラインフェーズの解析に基づき、**13回**ページフォールトが発生すると確実にここに到達している事を保証する事が出来る

Foreshadow攻撃例 – LEへの攻撃（9/10）

- この時点で**起動キー**は直近で使用されている関係上**L1Dキャッシュ**に**存在**しているため、**リターン**における**ターミナルフォールト**に伴う**過渡的実行**から、**Foreshadowによる起動キーの抽出**に**成功**している

- 実際に抽出した**起動キー**を用いて、**LEを迂回する**形で**EINITTOKEN**を**偽造し不正に起動させる事に成功**している
  - LEでEGETKEYにより起動キーの生成を行わない限り、KeyIDはひたすら同一のままである為、鍵の鮮度（Freshness）検証を排除できる

Foreshadow攻撃例 – LEへの攻撃（10/10）

- ただし、これをやった所で基本的には**Intelの利権の塊**である**ライセンス管理**を**迂回できるだけ**であるため、**SGX自体**の**セキュリティ**に**及ぼす影響**は**小さい**
  - LEが散々批判され、遂にはDeprecatedになった所以でもある

- ただし、**PKへのアクセス権**を持つ**ユーザEnclaveを起動**させる事が出来てしまうという点については**懸念すべき**である
  - 実際にはPKの導出にはMRSIGNERが必要であり、これを得るにはIntelのプライベート署名鍵が必要になるため、実質的には**PKの導出は不可能に近い**

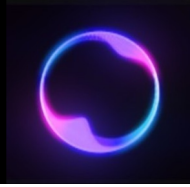
Foreshadow攻撃例 – QEへの攻撃

- 同様にして、**QEにおけるEGETKEY**に対して**Foreshadow攻撃**を仕掛ける事で、**レポートキー**や**PSK**の抽出にも成功している
  - PSKが抽出できると**Attestationキーをアンシーリング**出来るため、**Attestationキーの抽出**にも**成功**している

- **SGX Fail**の論文でも、**PowerDVD**に対し**Foreshadow攻撃**を仕掛け**Attestationキーを抽出**する事で攻撃を実現させたのはSGX Failのセクションで述べた通り

- **レポートキー**や**Attestationキー**の**漏洩**によって発生する**影響とその対策**についてもSGX Failのセクションで解説済み

本セクションのまとめ



- Controlled-Channel攻撃について解説を行い、実際にEnclaveのページアクセス権の拒絶に伴う挙動からサイドチャネル的に秘密情報を推測するPoCコードの実装を行った

- ハードウェア実装上の初期化不良バグを継いた攻撃であるÆPIC Leakと、SGXに対する初のMeltdown型攻撃であるForeshadow攻撃について説明した

- ÆPIC LeakやForeshadowは極めて高度な攻撃であるため、実践は全く必須ではない

参考文献（1/2）

[1]”Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems”, Yuanzhong Xu et al., https://ieeexplore.ieee.org/document/7163052

[2]”Intel SGX - Controlled-Channel Attacks解説”, 自己引用, https://qiita.com/Cliffford/items/f527fd210e3f7866e803

[3]”SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control”, Jo Van Bulck et al., https://jovanbulck.github.io/files/systex17-sgxstep.pdf

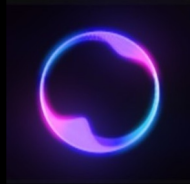
[4]”MDS: Microarchitectural Data Sampling”, 2023/7/20閲覧, https://mdsattacks.com/

[5]”ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture”, Pietro Borrello et al., https://aepicleak.com/aepicleak.pdf

[6]”AES key schedule - Wikipedia”, 2023/7/27閲覧, https://en.wikipedia.org/wiki/AES_key_schedule

[7]”How Stuff Gets eXposed”, 2023/7/27閲覧, https://sgx.fail/

参考文献（2/2）



[8]”Local APICについて - 睡分不足”, 2023/7/25閲覧, https://mmi.hatenablog.com/entry/2017/03/27/202656

[9]”What is the semantics for Super Queue and Line Fill buffers?”, 2023/7/25閲覧, https://stackoverflow.com/questions/45783251/what-is-the-semantics-for-super-queue-and-line-fill-buffers

[10]”SoK: SGX.Fail: How Stuff Gets eXposed, by Stephan van Schaik et al., https://sgx.fail/files/sgx.fail.pdf

[11]”FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution”, Jo Van Bulck et al., https://foreshadowattack.eu/foreshadow.pdf

[12]”Questions about launch_token and EINITTOKEN”, Intel, https://community.intel.com/t5/Intel-Software-Guard-Extensions/Questions-about-launch-token-and-EINITTOKEN/td-p/1094870 (魚拓: https://archive.is/vp6hL)