10. SGX攻擊編③

Ao Sakurai

2023年度セキュリティキャンプ全国大会 L5 - TEEの活用と攻撃実践ゼミ

本セクションの目標



SGXに対する攻撃の中でも最新の一角を担っている、
 「Foreshadow (L1 Terminal Fault) 」「LVI (Load Value Injection) 」「ÆPIC Leak」の3つについて解説する

これらの攻撃は実践するには極めて高度であるため、あくまでも 本ゼミでは解説を行うに留める

Foreshadow (L1 Terminal Fault)

キャッシュ階層



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

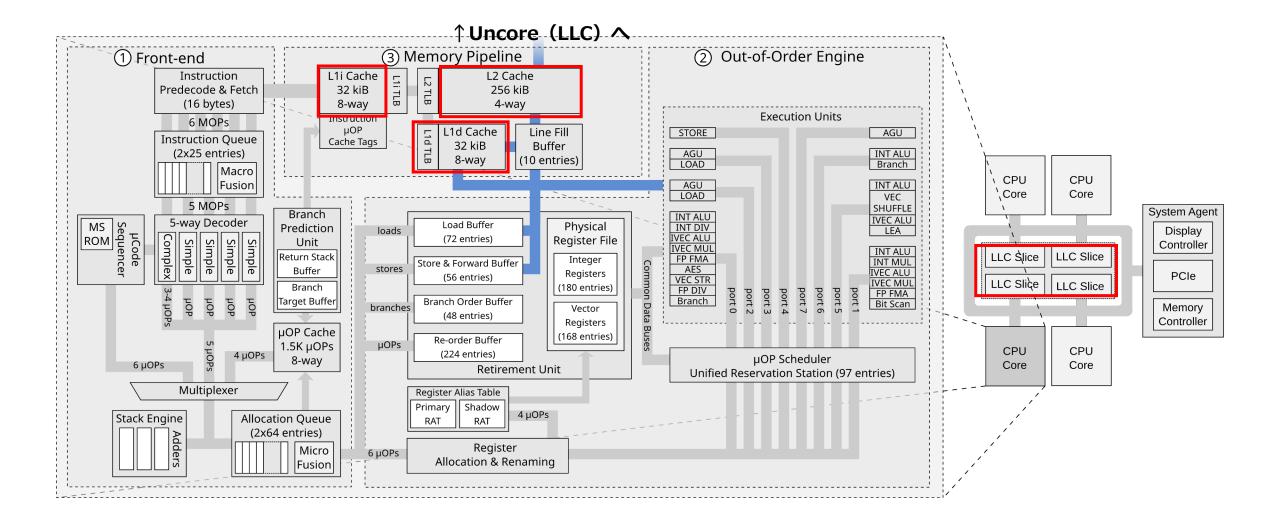
最近のCPUでは、CPUの主要部分に近い順にL1, L2, L3キャッシュの3つのキャッシュにより成り立っている

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



・以下の図はSkylake CPUの構造を示すものである([2]より引用)



Foreshadow攻擊(1/4)



Enclave内のデータも、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)



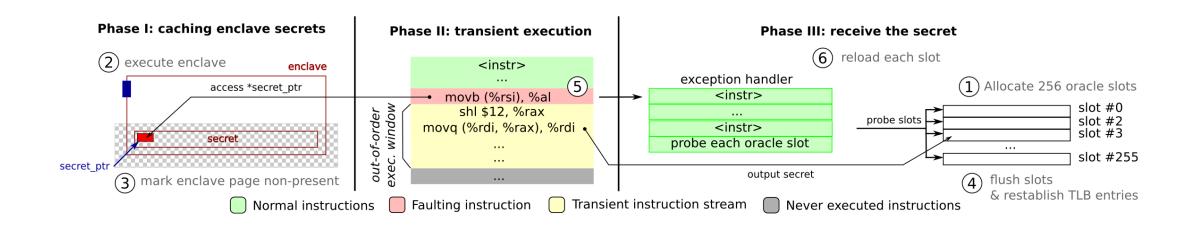
• ここではまず、Foreshadowの概念実証的な攻撃コードについて 説明を行う

- 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([1]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

- ・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、 起動十一(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
 - そもそも起動が不許可であったり、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

Foreshadow攻撃例 – LEへの攻撃(4/10)



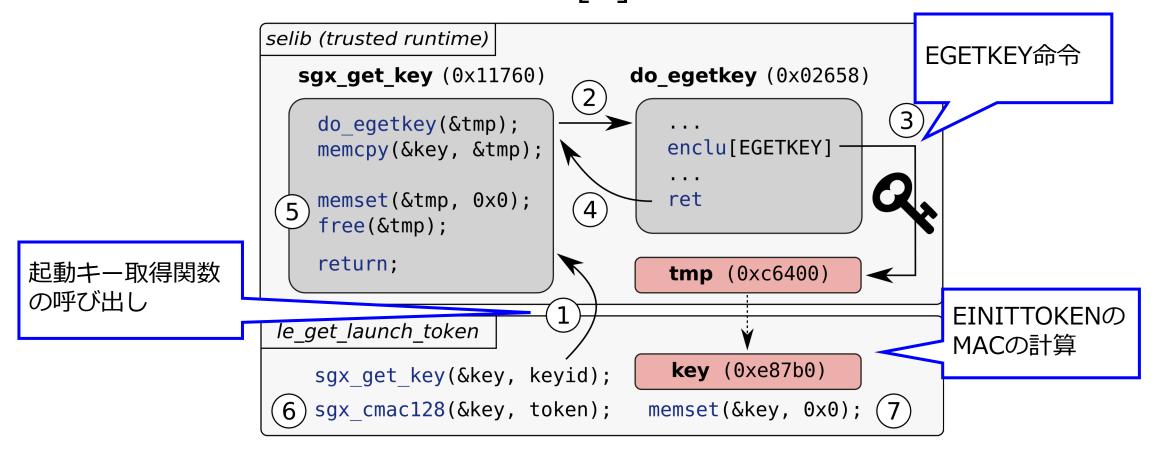
- もしForeshadow攻撃により起動キーを漏洩できた場合、完全に LEを迂回してEINITTOKENを偽造する事が出来てしまう
 - ・本来起動許可されていないEnclaveを起動したり、PKへのアクセス権を 不正に付与してEnclaveを起動させたり出来てしまう

- 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([1]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
 - Plundervoltの解説でも述べた通り、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

Load Value Injection (LVI)

Load Value Injection (1/4)



- ForeshadowはMeltdown型の攻撃であり、µ-Archバッファである
 L1Dから過渡的実行において秘密情報を漏洩させていた
 - その後、過渡的実行の結果が棄却されても後から観測できるように、 キャッシュに痕跡を残しサイドチャネル攻撃で抽出する
- これに対し、過渡的実行においてMeltdown挙動によりµ-Arch バッファから流れ出たデータを、その過渡的命令に対する注入に 使う攻撃がLoad Value Injection (LVI) である



Load Value Injection (2/4)



- 名前の通り、ロード命令(あるいはロードマイクロ命令)においてフォールトまたはマイクロコードアシストを発生させ、それに伴うそのロード命令の過渡的実行に対しµ-Archバッファから値をMeltdown的に注入(インジェクション)する攻撃
 - μ-Archバッファは予め攻撃に使う値で**汚染**(ポイズニング)しておく
 - フォールトとしてはページフォールト等が挙げられる
 - マイクロコードアシスト:普通は滅多に発生しないような状況に遭遇した際に、マイクロコードがそれを対処する動作
- ・広義にはPlundervolt同様**故障注入攻撃の一種**でもあり、実際に そのように振る舞う事も出来る

Load Value Injection (3/4)



 本来秘密情報の漏洩を行うMeltdown挙動をロード命令への値の 注入に転用しているため、逆Meltdown攻撃とも、あるいは MeltdownをSpectre的な注入に繋げるという意味で二者の融合型 攻撃とも表現する事が出来る

μ -Arch	Methodology Buffer	Leakage	Injection 📈
Prediction history	PHT	BranchScope [15], Bluethunder [24]	Spectre-PHT [38]
	BTB	SBPA [1], BranchShadow [40]	Spectre-BTB [38]
	RSB	Hyper-Channel [8]	Spectre-RSB [39, 44]
	STL	_	Spectre-STL [23]
Program data	L1D	Meltdown [42]	LVI-NULL
	L1D	Foreshadow [61]	LVI-L1D
	FPU	LazyFP [57]	LVI-FPU
	SB	Fallout [9]	LVI-SB
	LFB/LP	ZombieLoad [53], RIDL [67]	LVI-LFB/LP

LVIは、µ-Archバッファからの値の注入という新しい攻撃を 実現するものである(図は[6]より引用)

Load Value Injection (4/4)



・μ-Archバッファを**秘密情報の存在するアドレス**で汚染し、 過渡的実行でそのアドレスを注入し直接**キャッシュに秘密依存の 痕跡を残す**LVI手法を**ユニバーサルリードガジェット法**と呼ぶ

- 一方、µ-Archバッファを攻撃者の選んだ攻撃コードのアドレスで 汚染し、ret等におけるフォールトロードに伴う過渡的実行で そのアドレスを注入し攻撃コードに誘導するLVI手法を 制御フローリダイレクトガジェット法と呼ぶ
 - 誘導後に攻撃コードで秘密情報を漏洩させるような処理を行う

LVIに悪用できるµ-Archバッファ(1/3)



■L1Dキャッシュ

Foreshadowで説明した通り。

■ラインフィルバッファ(LFB)

L1Dに対して、他のキャッシュやメインメモリとのインタフェースとして機能するバッファ。

L1Dがキャッシュミスした場合、このLFBを介してより上位のキャッシュやメモリからデータが供給される[7]

■ロードポート (LP)

メモリやI/Oからのロードを行うポート。

LVIに悪用できるµ-Archバッファ(2/3)



■ストアバッファ (SB)

未処理のストアデータとアドレスを追跡(保持)するバッファ。 準備・状況が整ったら、インオーダーで実際にストアを行う。

実際のストア処理の完了を待つ代わりにこのバッファに一時的に保持させておく事で、命令パイプライン自体やアウトオブオーダー実行の高速化を図る事が出来る。**ストア操作による汚染が可能である。**

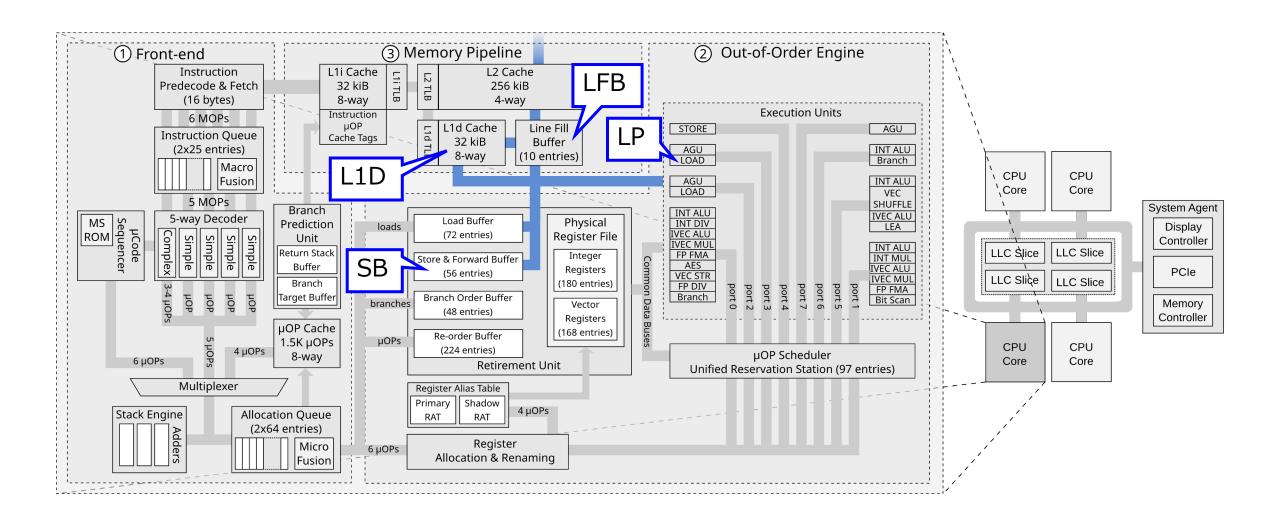
■浮動小数点演算処理装置(FPU)

その名の通り、浮動小数点演算を専門に行う、CPUに内蔵されている ユニット。

LVIに悪用できるµ-Archバッファ(3/3)



• CPU構造を示す図を再掲する([2]より引用)



LVIの攻撃力(1/4)



- ・攻撃対象のドメイン内(例:ユーザ空間であるEnclave)で攻撃の ほとんどが完結するという特徴がある
 - よって、Foreshadow等と異なり、攻撃対象のコードにおいてLVIを発生させる前提であり、Foreshadowのように攻撃者が自前のコードを用いて攻撃を発動させる事は原則想定されていない

・よって、**コンテキストスイッチ**(例:ユーザ→カーネルへの 切り替え)時にµ-Archバッファをフラッシュするような、従来の Meltdown型攻撃への対策はLVIには効かない

LVIの攻撃力 (2/4)



 Spectre攻撃に対する対策としては、メモリ曖昧性解消機能(*)の 無効化(Spectre-STLの場合)や、分岐予測器に汚染に対する耐性を 付与するような対策が取られている

(*)英語でmemory disambiguation. 詳細は省略

しかし、LVIはMeltdown的な挙動によりµ-Archバッファから漏洩した値を後続の命令に対して直接注入するため、分岐予測関係を補強する上記のようなSpectre対策は根本的に無意味である

LVIの攻撃力(3/4)



- ・また、Spectreについてはコード内で過渡的実行攻撃が発生しそうな 部分にIfence命令を挿入する事で対策する方法も有名である
 - Ifence命令: この命令が挿入された場所より後の命令が、Ifence命令よりも 先にアウトオブオーダー実行される事を防ぐための命令
- ・LVIでも後続の命令を過渡的に実行する事を抑止するためにIfenceが 有効であるが、挿入すべき対象があまりにも多く(例:ret命令)、 全て対応しているとかなりのオーバーヘッドとなる

しかも、性質上軽減策によるIfenceの導入が困難なロード命令も 存在するため、完全な対応は不可能に近い

LVIの攻撃力 (4/4)



結局の所、ロード命令においてフォールトやアシストが発生した際に 過渡的実行が発生しないようCPUのシリコンレベルで対策をしない 限り、LVIの根絶は限りなく無理に等しい

- 一方で、Meltdown挙動を発生させ、それにより注入された値を 後続の過渡的命令に使わせるという極めて難易度の高い攻撃であり、 攻撃の実用性は低いと言わざるを得ない
- このように、攻撃力というよりは攻撃可能範囲が極めて広いという 点が、LVIの厄介な性質である

LVIのPoC攻撃(1/3)



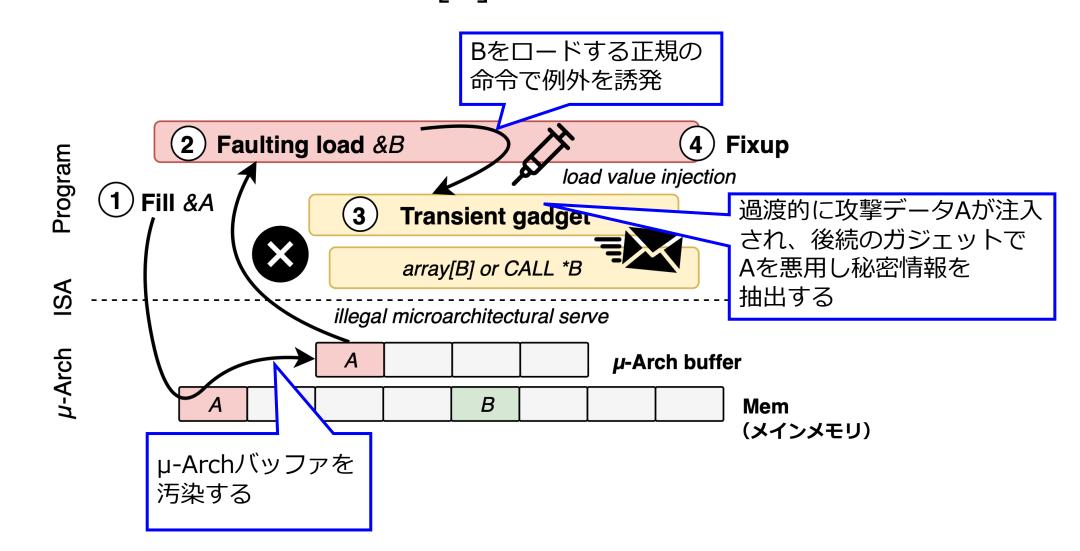
• Foreshadowの時と同様、まずはLVIの概念実証的な攻撃コードについて説明を行う

- ・LVIは、主に以下の3フェーズにより構成される:
 - フェーズI (P1): µ-Archバッファの汚染
 - ・フェーズII(P2):ロードにおけるフォールト/アシストの誘発
 - フェーズIII (P3): ガジェット(攻撃に利用可能なコード)ベースでの 秘密情報の転送(漏洩)

LVIのPoC攻撃(2/3)



• LVIの概要図は以下の通り(図は[6]より引用)



LVIのPoC攻撃(3/3)



・攻撃対象マシンで動作する以下のコードに対してLVI攻撃を行い、 秘密情報を抽出する事を考える:

```
void call_victim(size_t untrusted_arg) {
    *arg_copy = untrusted_arg;
    array[**trusted_ptr * 4096];
}
```

LVIのPoC攻撃 – P1ガジェット



- 2行目の*arg_copy = untrusted_arg;により、64ビット (=ポインタのサイズ)の信頼不可能な値(攻撃に使用する値)を 信頼可能なメモリ(Enclave内のスタック等)にコピーしている
- この動作によりメモリへのストアが発生するため、ストアバッファを 攻撃に使用する(=注入する)値で汚染できる
- よって、この2行目のコードはµ-Archバッファを汚染するための P1ガジェットである事になる
 - arg_copyはコピーによるSBの汚染を発生させるためだけに使ったので、 これ以降は登場しない

LVIのPoC攻撃 – P2ガジェット(1/4)



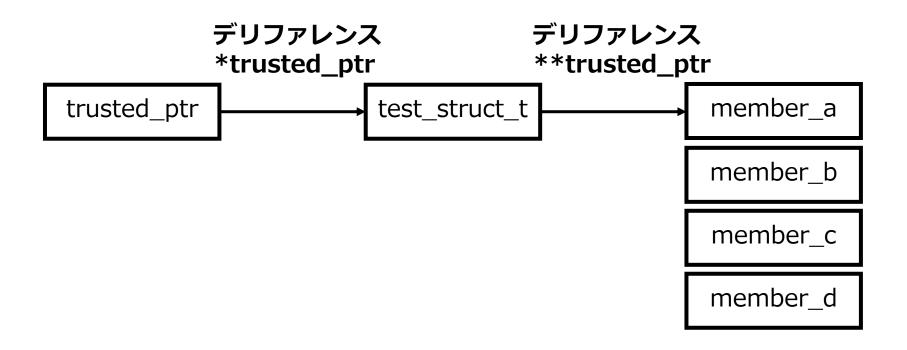
3行目の**trusted_ptrは**二重ポインタ**であり、例えば(動的に確保した)構造体へのポインタ等が具体例として挙げられる

```
uint8_t *test_st = new uint8_t[sizeof(test_struct_t)]();
      uint8_t **trusted_ptr = &test_st;
参照
                                          参照
*trusted ptr
                                          **trusted_ptr
      typedef struct {
          uint8_t member_a;
          uint8_t member_b;
          uint8_t member_c;
          uint8_t member_d;
       test_struct_t;
```

LVIのPoC攻撃 – P2ガジェット(2/4)



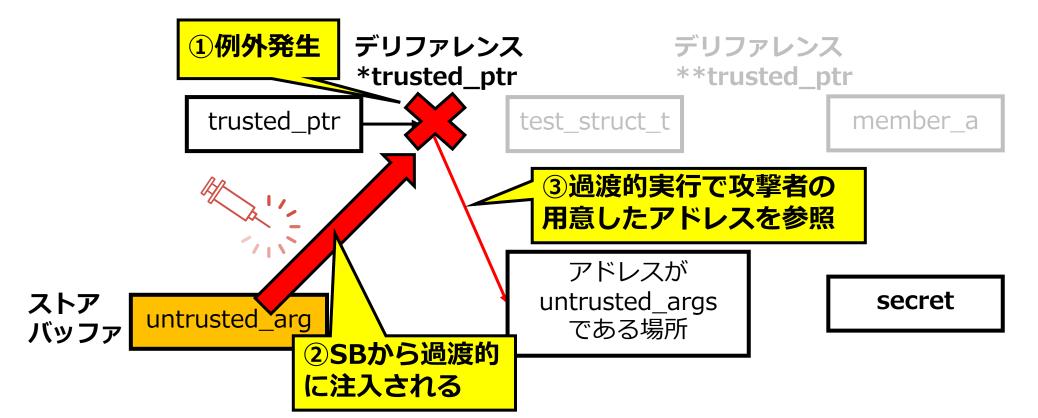
- 二重ポインタtrusted_ptrについて、*trusted_ptrのように1段階の 参照先取得(デリファレンス)を行うと、本来は構造体の 先頭アドレスが手に入る



LVIのPoC攻撃 – P2ガジェット (3/4)



 ここで、1段階のデリファレンスである*trusted_ptrにおいて フォールトまたはアシストを発生させる。すると、それに伴う 投機的実行において、SBから流れ込んできた値がデリファレンスに おける参照先として後続の過渡的命令で過渡的に使用されてしまう



LVIのPoC攻撃 – P2ガジェット(4/4)



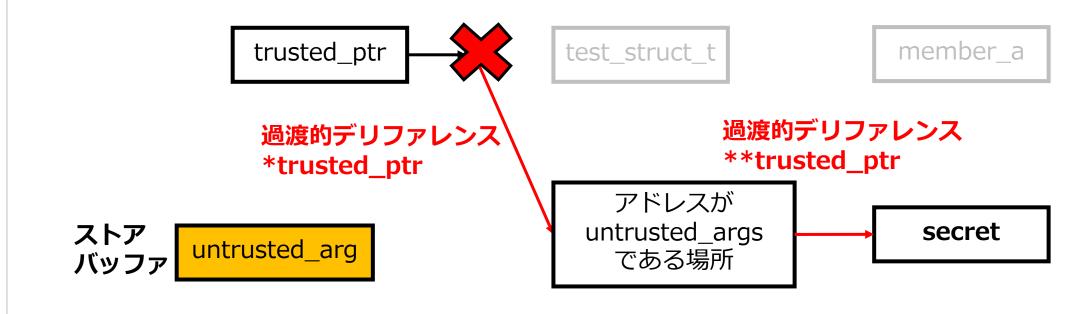
この1段階目のデリファレンスでフォールトまたはアシストを発生 させて過渡的実行を誘発できるため、このデリファレンスは P2ガジェットであると見なす事が出来る

- フォールトまたはアシストを発生させるには、何度か登場している mprotectを用いたり、ページテーブルエントリ(PTE)を 直接改竄する等の様々な手法が利用可能である
- ストアバッファからの値の注入を誘発しているので、このPoC攻撃は 後述のLVI-SBというバリアントに属する事が分かる

LVIのPoC攻撃 – P3ガジェット(1/2)



- 2段階目のデリファレンスは、注入されたuntrusted_argsをベースに実施されるため、untrusted_argsの場所に存在する秘密情報をフェッチしてしまう
 - ちなみに、過渡的実行が始まり、アーキテクチャの処理が追いついて棄却 されるまでの攻撃可能時間を過渡的実行ウィンドウ(Transient Window) という



LVIのPoC攻撃 – P3ガジェット (2/2)



- ・攻撃対象コードの3行目において、 監視用配列arrayに対して array[**trusted_ptr * 4096];のように秘密バイト依存の ページアクセスを行っている
- ・ここまで来れば**Foreshadowの時と同様**、過渡的実行が棄却された 後に**キャッシュに痕跡が残る**ため、FLUSH+RELOAD**キャッシュ サイドチャネル攻撃で秘密バイトを抽出**できてしまう

このPoC攻撃では、2段階目のデリファレンスから監視用配列への 秘密依存のアクセスまでがP3ガジェットである

LVIのバリアント



・ロード命令に対する注入をどこから行うかに応じて、LVIにはいくつかのバリアント(変種・種類)が存在する

- 本ゼミでは、原論文に倣い以下のLVIバリアントについて解説する:
 - LVI-L1D
 - LVI-SB
 - LVI-NULL

LVI-L1D (1/2)



その名の通りL1Dから値の注入を行うLVIバリアント

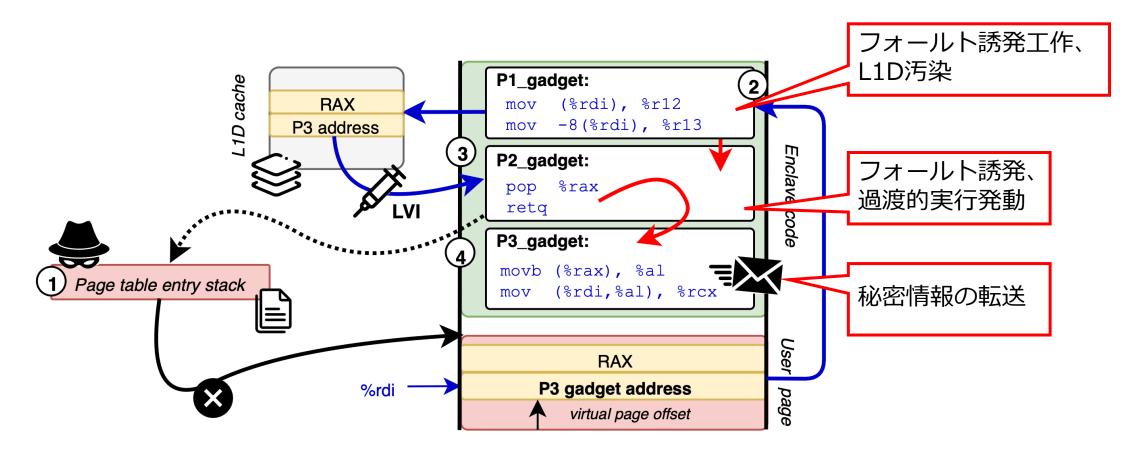
- L1Dからの漏洩を注入に転用する事から、LVI-L1Dは **逆Foreshadow攻撃**であると捉える事も出来る
 - 恐らくこの世に存在するSGX攻撃の中でも頂点に君臨するレベルの複雑度

• Foreshadowに対する対策が適用されている環境では、LVI-L1Dによる攻撃を成立させる事は出来ない

LVI-L1D (2/2)



• LVI-L1Dの概要図は以下の通り([6]より引用):



アセンブリはAT&T形式([命令] <ソース> <宛先>)で書かれているので注意

LVI-L1D - P1ガジェット (1/3)



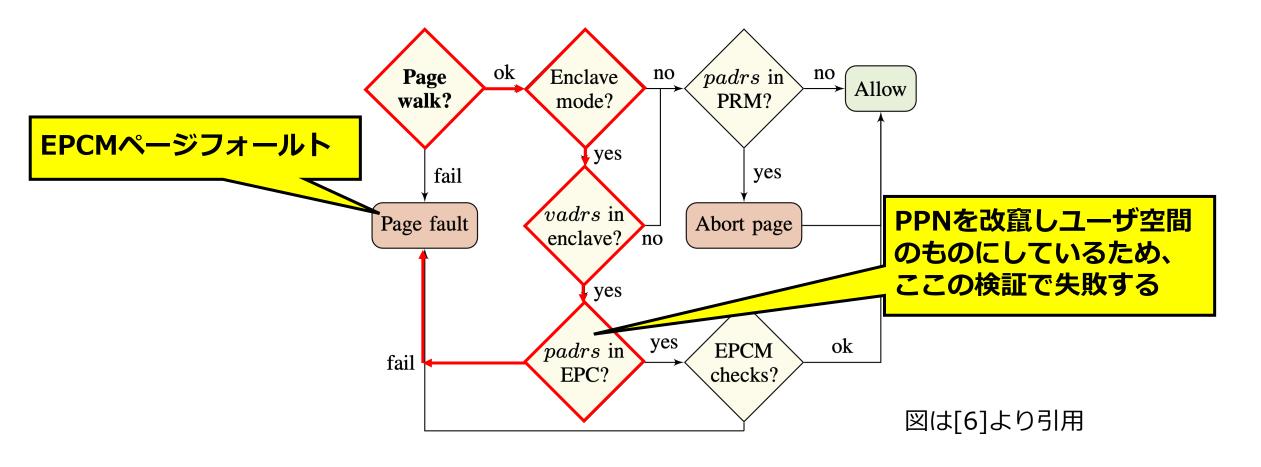
ページフォールトシーケンス手法やSGX-Stepを用いる事で、 Enclaveの実行をP1ガジェットの直前まで正確に進める

- その後、P2_gadgetのretq命令がロードするスタックページの PTE内のPPN(物理ページ番号)を、P3ガジェットのアドレス値を 格納するユーザ空間上のページ(以下、ページUと呼ぶ)のものに 改竄する(前図①)
 - アドレス値はuintptr_tをイメージすると分かりやすい

LVI-L1D - P1ガジェット (2/3)



 このPPNの改竄により、P2におけるretqの実行時に、Enclaveの 正常な範囲の外へ飛ぼうとした事による、EPCMページフォールトと 呼ばれる特殊なターミナルフォールトが発生する



LVI-L1D - P1ガジェット (3/3)



その後、攻撃対象のEnclave内に存在する、例えばout属性のOCALL引数(=攻撃者の用意した値で、P3ガジェットのアドレス値)をコピーするようなP1ガジェットにより、ページU上の攻撃用の値をL1Dにキャッシュさせる(概要図②)

前述の図においては、ページUの保持する値(=P3ガジェットのアドレス値)を%r12及び%r13レジスタにmovする事で
 L1Dへのキャッシュを行っている

LVI-L1D - P2ガジェット (1/2)



- P2ではまず、P2の載っているEnclave内関数からの戻り値 (前図のシナリオではこれが秘密情報である)をスタックから %raxにポップしている
 - 前図のP1・P2ガジェットは、別のEnclave内関数から呼び出されている Enclave内関数であるのが前提である
- P1でのPPN改竄の仕込みにより、P2_gadgetのretqが実行されると、Enclaveの仮想ページに対しEnclave外のページUの物理ページが割り当てられているため、EPCMページフォールトが発生する(前図③)
 - ここでEPCMページフォールトに伴う**過渡的実行が発生**する

LVI-L1D - P2ガジェット (2/2)



しかし、これに伴う過渡的実行においてページUのアドレスをベースにL1Dへの問い合わせが行われ、P1でページUはキャッシュ済みであるため、過渡的なretqのジャンプ先としてページUの値(=P3ガジェットのアドレス)が使用されてしまう

LVI-L1D - P3ガジェット (1/2)



- P2により同一Enclave内であればどこにでも過渡的に制御フローを 転送できるため、Enclave内に存在する、攻撃者の選択した P3ガジェットに転送させる事が出来る
 - P3ガジェットの位置の指定は、ページUが保持する、P3ガジェットのアドレス値により行う
- 後はそのP3ガジェットを用いてキャッシュに痕跡を残し、 過渡的実行リタイア後にキャッシュサイドチャネル攻撃で秘密バイト を観測出来てしまう

LVI-L1D - P3ガジェット (2/2)



前図のP3ガジェット(④)では、秘密バイトである戻り値をP2でポップして格納した%raxを、%rdiに対するインデックスとして使用し、そのアクセスにより秘密バイトの痕跡を残している

- 元々%rdiにはP3ガジェットのアドレス値が格納されていたが、 P3フェーズに入ると%rdiに何が入っていたかは関係ない
 - 言わばここでは%rdiを再利用しているだけであり、監視用配列として 使用しているだけで元々何が入っていたかは既にこの時点で無意味である

LVI-SB (1/3)



・前述のLVIのPoC攻撃も属しているバリアントのLVI攻撃であり、 ストアバッファからの値の注入を悪用する攻撃

 Fallout攻撃[8]の論文により、SBに対するMeltdown挙動を 取る場合、ロード命令におけるページオフセット(対象アドレス 下位12bit)が最近の未処理のストアのそれと一致しなければ ならない事が判明している

LVI-SB (2/3)



・攻撃対象として、**Edger8r**が生成するエッジコードの以下の例を取り上げる:

```
; %rbx: user-controlled argument ptr (outside enclave)
sgx_my_sum_bridge:
    call my_sum; compute 0x10(%rbx) + 0x8(%rbx)
    mov %rax,(%rbx); P1: store sum to user address
    xor %eax,%eax
    pop %rbx
    ret; P2: load from trusted stack
```

LVI-SB (3/3)



Enclave外のポインタ(例:配列の先頭ポインタ)を引数として 受け取り、Enclave内でそのポインタ先が含む2つの値を加算して 引数経由でリターンするECALL関数についてのエッジコードである

・加算結果をストアバッファに格納させ、8行目のret命令で 過渡的実行を発動させる事で、SBからの注入によりリターン先 として攻撃者の選択するP3ガジェットに飛ぶように仕向ける攻撃を 考える

LVI-SB - P1ガジェット



- 3行目の「…」部では、結果の返却にも用いる引数ポインタ ([in, out]属性をイメージすると分かりやすい)がEnclave外に 存在しているかを確認しているだけである
 - ・よって、攻撃者が用意した引数の値がEdger8rによって阻害される事はない
- ・結果として、攻撃者はこの引数の加算値(=P2でret先として 過渡的にSBから注入されるアドレス値になる)をEnclave外で 任意に設定(改竄)し、前述のページオフセット一致の制約を 容易に満たす事が出来る
- ・この性質を利用し、攻撃が上手く行くように引数を渡し、my_sum 関数での加算結果をSBに格納させればP1は完了

LVI-SB - P2ガジェット



- 4行目のcall my_sumの後にEnclaveを中断し、8行目のretで参照 する事になるEnclaveスタックのページでのフォールトやアシストを 誘発する
 - ・論文中では、ここではこのスタックページに対応するPTEのAccessedビット かスーパーバイザビットをクリアする事でこれを実現すると述べている
 - ただし、今まで通りPresentビットをクリアするのでは不可能であるとは書かれておらず、このケースに限ってPresentビットクリアが通用しない理由も無さそうであるため、Presentビットによる誘発も可能そうである
- 8行目で実際にret命令が呼ばれると、フォールトまたはアシストが発生し、過渡的実行でSBから引数同士の加算値(=P3ガジェットの位置を指すように先程している)が注入される

LVI-SB - P3ガジェット



あとはLVI-L1Dの時と同様、任意のP3ガジェットに制御フローを リダイレクト出来ているため、それを用いて秘密情報の漏洩を行える

•制御フローリダイレクトに限らず、ユニバーサルリードガジェット法的にLVI-SB攻撃を行う事も勿論可能である

LVI-NULL (1/7)



- Meltdownへの対策を行っている直近の世代のCPUでは、過渡的実行でMeltdown的に漏洩する値をダミーの0x00に強制的に置き換える事で、秘密情報が漏洩する事を防ぐようにしている
 - MSRのIA32_ARCH_CAPABILITIESアドレスのRDCL_NOビットが1であればMeltdown対策済みである
 - RDCLは**Rogue Data Cache Load**の略で、**Meltdownの正式名称**。NOは 文字通り否定のnoの意

・しかし、**過渡的実行に0が注入される**という**挙動自体**を悪用する LVI攻撃も実行可能であり、これがLVI-NULLである

LVI-NULL (2/7)



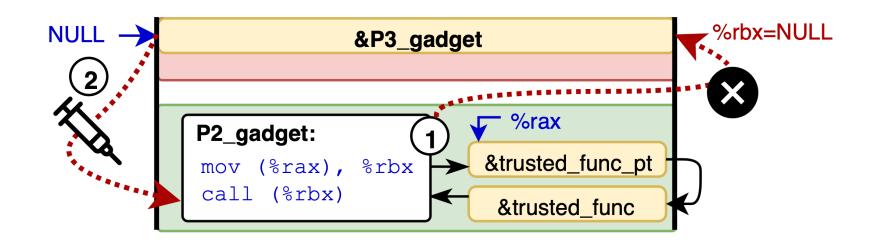
- OSや環境にもよるが、root権限を持つ攻撃者は仮想アドレスNull (=アドレス0)に任意のメモリページをマッピングする事が出来る
 - 手っ取り早い方法として、PTE内に登録されている仮想アドレスを0にしてしまえば良い

 よって、Meltdown対策済みCPUによる、過渡的実行における0値の 転送(注入)を悪用し、仮想アドレス0に用意したP3ガジェットに 制御を転送する攻撃を例として考える

LVI-NULL (3/7)



ここでのLVI-NULL攻撃の概要図は以下の通り([6]より引用):



LVI-NULL (4/7)



- このシナリオでは、関数の二重ポインタ(**ptrのように デリファレンスする事で関数にアクセスできるようなポインタ)に おいてLVIを行う事を考える
 - 関数の二重ポインタの具体的な実例として、動的に確保した構造体のような ヒープオブジェクトに含まれる関数ポインタが挙げられる

・二重ポインタに対して攻撃するという意味では、LVIのPoC攻撃の 説明で示したLVI-SBのケースと若干似ている

LVI-NULL (5/7)

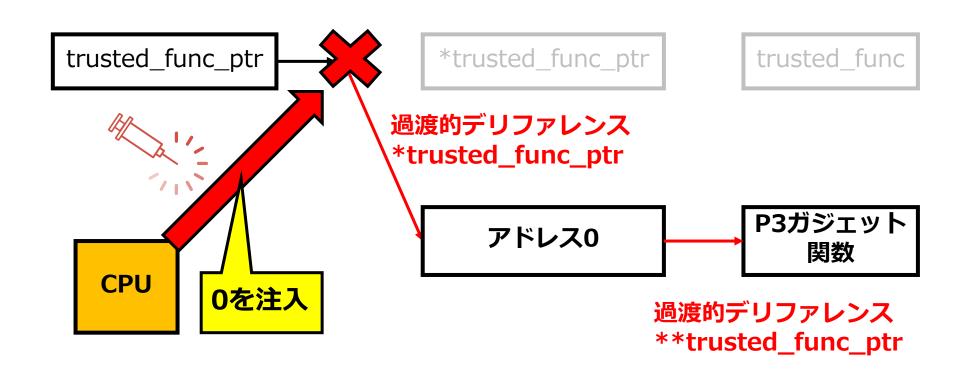


- ・この関数の二重ポインタの**第一段階のデリファレンス**でフォールトや アシストを誘発する事で、**過渡的実行を発生**させる
- ・この時、Meltdown耐性のあるCPUは過渡的実行に**ダミー値0**を 転送し、結果的に**過渡的**な**第一段階のデリファレンス**で**アドレス0**を 使用してしまう
- 結果として、第2段階のデリファレンスでは攻撃者が用意した アドレス0をベースアドレスとした場所にある不正な関数ポインタを 過渡的に取得してしまい、それによりP3ガジェットに制御転送 されてしまう

LVI-NULL (6/7)



• このLVI-NULLの実行の様子を示した図は以下の通り:



LVI-NULL (7/7)



- ここで1段階のみの関数ポインタを用いると、過渡的な関数呼び出しが上手く行かないらしい
 - 詳細は不明だが、一度過渡的にEnclave外ページをデリファレンスしてから その上のEnclave外関数を呼ぶ事は出来るが、直接Enclave外関数を呼ぶのは 過渡的実行上でも不可能であると推測できる
 - デリファレンス先ページを実行不可能とマークする等して時間を稼ぎ、 その間にアドレス0に再配置可能なEnclaveイメージをロードすれば 1段階のみの関数ポインタでも攻撃が成立する可能性はある
- LVI-NULLの場合、他のバリアントと比べても過渡的実行ウィンドウ が小さいため、実際に有効な攻撃を成功裏に行うのは極めて難しいと Intelは主張している[9]

LVI攻撃例 - AES-NIへの攻撃(1/8)



- LVIを利用したより実践的な攻撃として、**AES-NIを利用する**ような Enclaveに対し**LVI-NULL**攻撃を仕掛ける事で、**共通鍵を抽出**する ような**故障注入攻**撃を行うケースを考える
 - AES-NI: AES暗号の暗号化及び復号の高速化を目的に実装されている、 x86の拡張命令
- 既知暗号文攻撃(暗号文のみを知った状態で秘密を解読する攻撃)のシナリオを前提とし、正しい暗号文を復号する処理に対して 故障注入攻撃を行う事を考える

前述のLVI-NULLの例が制御フローの改竄を行っていたのに対し、 この例はある意味Plundervoltに近い故障注入攻撃を行う点で かなり毛色が異なる

LVI攻撃例 – AES-NIへの攻撃(2/8)



AES暗号では、暗号文を128bit (16バイト)のプロックという単位に分割し、さらに各ブロックを1マス8bit (1バイト)とした4×4の行列にする

- そして、この4×4行列に対し、ある一連の処理をAESの仕様で 定められているラウンド数分だけ繰り返す(鍵伸長処理)
 - ・ 鍵長が128bitである場合はこのラウンド数は10である

LVI攻撃例 - AES-NIへの攻撃 (3/8)



- 一連の処理とは、暗号化の場合は順に以下の通り:
 - SubBytes: S-Boxという置換表を参照するByte単位の置換
 - **ShiftRows**: 4×4行列のn行目を(n-1)マスだけ左側にずらす。左端を超えるようなマスは右端に行くようにする(回転)
 - MixColumns:列単位の処理。数式が長いので省くが、XORをベースとした 演算(CBCにおけるXOR処理とは別物)
 - AddRoundKey: state(処理中の4×4行列。上記の処理を適用した状態の途中段階の行列)とラウンド鍵で、列ごとにXORを取る
- 最終ラウンドのみ、MixColumnsは実行されない

LVI攻撃例 – AES-NIへの攻撃(4/8)



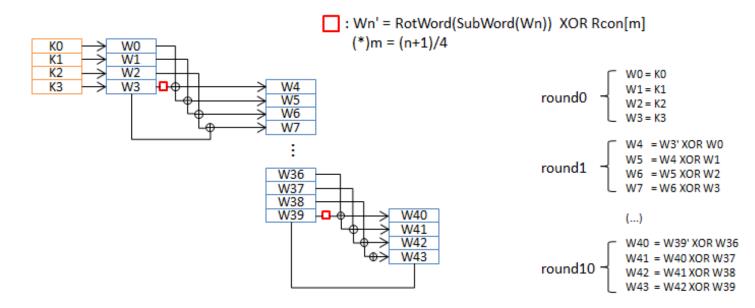
- 復号の場合は以下の通り:
 - **InvShiftRows**:暗号化時のShiftRowsの右回転バージョン
 - **InvSubBytes**: Inverse S-Boxを参照した、SubBytesの逆置換
 - ・AddRoundKey:暗号化時と同様
 - InvMixColumns: これも列単位の処理で、数式が複雑なので詳細は割愛
- 最終ラウンドのみ、InvMixColumnsは実行されない

LVI攻撃例 – AES-NIへの攻撃(5/8)



ここで、ラウンド鍵はAESの共通鍵からラウンド分だけそれぞれ 導出されるもので、以下のようにして導出される (図は[10]より引用)

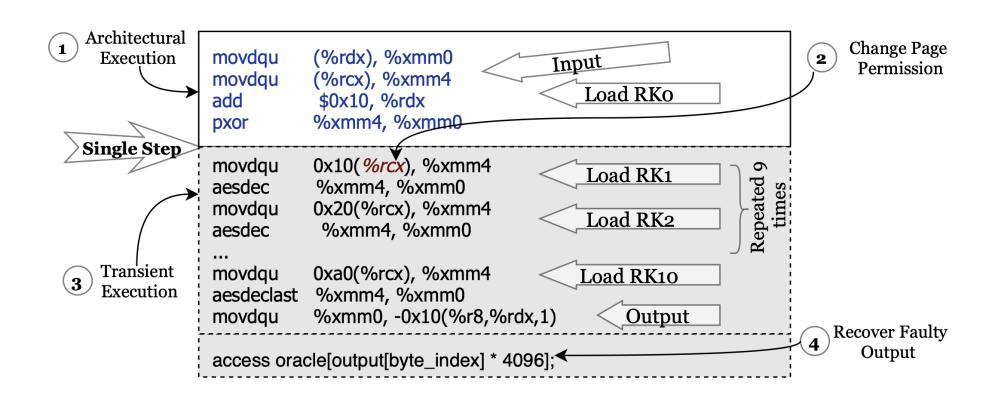
AES-128



LVI攻撃例 - AES-NIへの攻撃(6/8)



AES-NIを利用するEnclaveに対するLVI-NULL攻撃の概要図は 以下の通り(図は[6]より引用):



LVI攻撃例 – AES-NIへの攻撃(7/8)



 まず、SGX-Stepを使用して最初(第0)のラウンドを実行した後に 攻撃対象のEnclaveに正確に割り込む

その後、ラウンド鍵が格納されたメモリページのアクセス権を剥奪 してからEnclaveを再開する

・続くラウンド処理で**ラウンド鍵へのアクセス**により**フォールトが発生** するため、最初以外のラウンドではMeltdown対策挙動により **オールゼロのラウンド鍵が過渡的に使用(LVI-NULL)**されてしまう

LVI攻撃例 – AES-NIへの攻撃(8/8)



- 第0ラウンド鍵以外のラウンド鍵として全ラウンドにてオールゼロの 故障した鍵で復号された、故障した平文をキャッシュに残す
- 「暗号文⊕第0ラウンド鍵⊕オールゼロ鍵=故障した平文」である ため、XORの性質より、
 - 「故障した平文⊕オールゼロ鍵=**暗号文⊕第0ラウンド鍵**」となる
 - ・暗号文は既知であるため、これにより第0ラウンド鍵が抽出できる
- 第0ラウンド鍵は前図の通り共通鍵そのものであるため、これにより 目当ての共通鍵を抽出できてしまう
 - 実際には単純なXORだけでなくAES特有の処理が挟まるため、AESの逆処理 を行う関数を用意して共通鍵の抽出を行う

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-Stepも、Local APICのAPICタイマを用いてシングルステップ実行を 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)



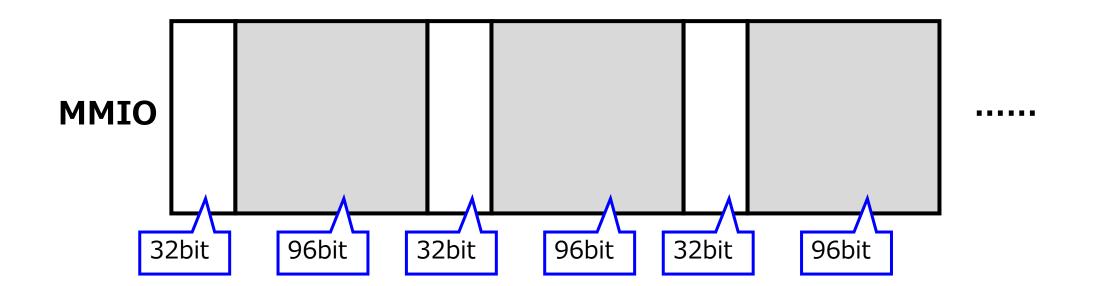
MMIOのベースアドレスはデフォルトでは物理アドレス 0xFEE00000に設定されているが、MSRのIA32_APIC_BASEの 値を変更する事でコアごとに対応するMMIOアドレスを変更できる

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



メモリマップト領域 パディング領域

APIC (5/5)



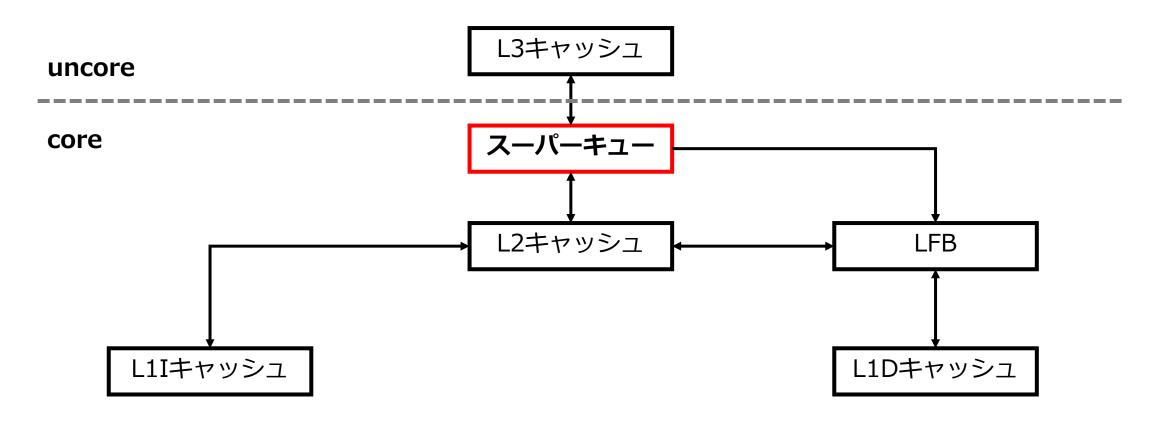
このパディング領域は文字通りパディング用であり、それ以外に何らかのアーキテクチャ的な定義がなされていない領域である

- 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(1/2)



 ForeshadowとLVIの説明では言及しなかったが、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が存在する



キャッシュ階層 (2/2)



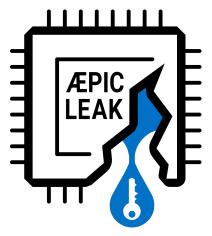
APICは、L3キャッシュからL2キャッシュに値を持ってくる際に、 このスーパーキューを使用する

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)



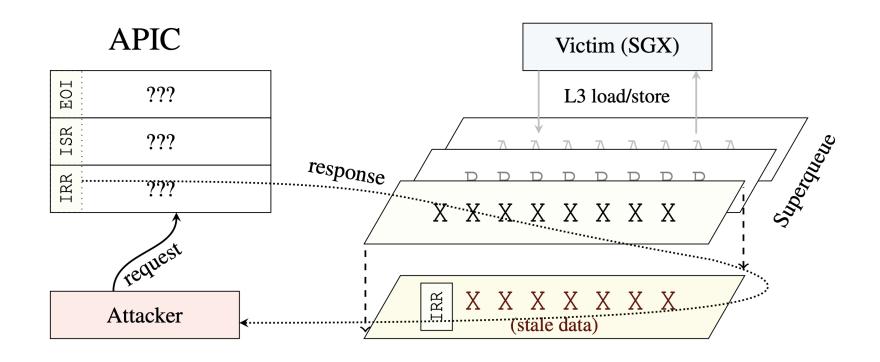
- Sunny Coveマイクロアーキテクチャを搭載するCPUにおいて、 xAPICのMMIOのパディング領域を読み取ると、スーパーキューに 残留している値が漏洩するバグがある事が判明した
 - Sunny Cove µ-Archを搭載するCPUとしては、第10~13世代CoreシリーズCPUや、第3世代Xeon-SP CPUが挙げられる
- ・このバグを悪用し、Enclave由来のスーパーキュー内の秘密情報を 漏洩させる攻撃がÆPIC Leakである
 - ・根本原因は、APICレジスタ(MMIO)の**パディング部分**が **正しく初期化されない**という**アーキテクチャ上のバグ**である
 - ・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([12]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)



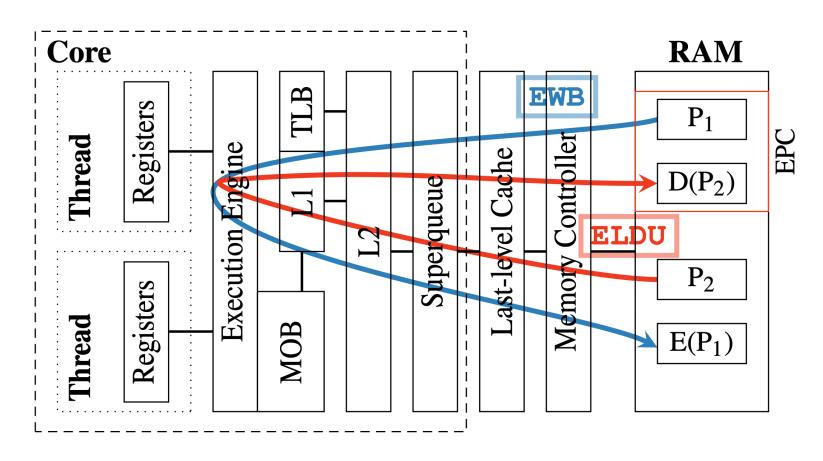
 Foreshadow攻撃でも使用されていた手法であるが、秘密情報が 載るEPCページをEWBとELDUでページ退避&ロードを繰り返す 事で、強制的に秘密情報の載るページの内容をキャッシュ階層 (スーパーキュー含む) に持ち込む事が出来る

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([12]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)

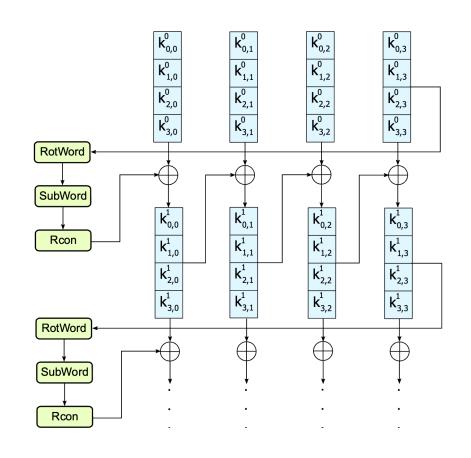


- ÆPIC Leakを仕掛けると、Nラウンド目とN+1ラウンド目の ラウンド鍵の、それぞれ後半3ワードが復元できる
 - 1ワード: サイズは4バイトで、AESブロックである4×4行列の1行 または1列に相当する。ワード換算では128bitは4ワードになる
 - ・ 先頭1ワードが抽出できないのは、前述の37.5%のリーク率が原因である

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[14]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(図は[14]より引用)
- 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)



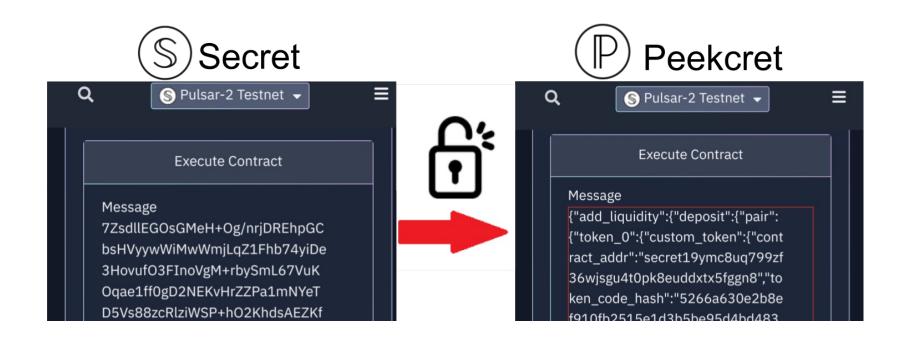
 これは即ち、Nラウンド鍵とN+1ラウンド鍵のそれぞれ第2ワード 同士をXORすれば、N+1ラウンド目の先頭ワードが復元でき、 N+1ラウンド鍵が完全に抽出できる事になる

- 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[14]より引用)



まとめ



• SGX攻撃の中でも最先端の一角を担っている、Foreshadow、LVI、ÆPIC Leakの3つの攻撃についてある程度詳細に解説した

• これらの攻撃を実践するのは非常に難易度が高いため、その実践は 完全にSGXを極めるのであればおすすめする

参考文献(1/3)



[1]"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

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

[3]"Intel SGX Explained", Victor Costan & Srinivas Devadas, https://eprint.iacr.org/2016/086.pdf

[4]"Overview on Signing and Whitelisting for Intel® Software Guard Extension (Intel® SGX) Enclaves", Intel,

https://www.intel.com/content/dam/develop/external/us/en/documents/overview-signing-whitelisting-intel-sgx-enclaves-737361.pdf

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

[6]"LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection", Jo Van Bulck et al., https://lviattack.eu/lvi.pdf

参考文献(2/3)



[7]"ZombieLoad: Cross-Privilege-Boundary Data Sampling", Michael Schwarz et al., https://zombieloadattack.com/zombieload.pdf

[8]"Fallout: Leaking Data on Meltdown-resistant CPUs", Claudio Canella et al., https://mdsattacks.com/files/fallout.pdf

[9]"Load Value Injection", Intel, https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/load-value-injection.html

[10]"AESを理解する", Qiita, 2023/7/25閲覧, https://qiita.com/tobira-code/items/152befa86bd515f67241

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

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

参考文献(3/3)



[13]"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

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

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

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