7. SGX Fail

Ao Sakurai

2025年度セキュリティキャンプ全国大会
L3 – TEEビルド&スクラップゼミ

本セクションの目標

- **SGXのクソ仕様を痛感する。**

- SGXやTEEを、実世界の様々なシナリオで応用する際に直面するさまざまな困難について解説する。

勘弁してくれSGX

ECALLによるバッファの転送（1/4）

- Enclave外のAppで**10MB**の**バッファ**を用意し、ECALLで
  その**バッファをEnclave内にロードする**プログラムを実装する
  - バッファの型は**uint8_t***型で良い
  - バッファはAppにおいてnew演算子等で確保する
  - バッファの内容は**全バイト'A'**とする。**memset**を用いると楽
  - EDLのポインタ方向属性を[in]にしてEnclaveに全バイト読み込む
  - Enclave設定XMLを編集し、**Enclaveヒープサイズ**を**10MB以上**
    （余裕を見て**20MB程度**）確保するようにする

- 実行はローカルで良い（RA不要）

ECALLによるバッファの転送（2/4）

- このタイミングでわざわざ出しているのだから上手く行くはずがない

- 恐らく**Enclave**が**正体不明の死**を迎え、ゼミで提供しているSGXステータス表示関数を使うと以下のように出るはずである：

```
Execute ECALL.


==========================================================================
SGX_ERROR_ENCLAVE_CRASHED
The enclave has crashed.
==========================================================================
```

ECALLによるバッファの転送（3/4）

- この事象は、**ECALLやOCALLの際、境界を跨いで引き渡すバッファを一度内部のスタックに格納する**という**暗黙の仕様**が存在する事により発生する
  - 当然開発者リファレンスに書かれていない

- このスタックは上限が**8MB**であるため、**それを超えるバッファ**を渡した瞬間に**Enclaveが即死**する

ECALLによるバッファの転送（4/4）

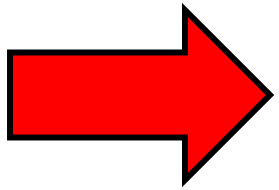
- このスタックサイズは**OSの設定に依存**するため、特権で**ulimitコマンド**を使用する事などで無制限には出来る
  - ただし、自作のSGXアプリを自分だけが使うのであればともかく、例えばサービスとしてSGX依存の機能を提供する場合、**スタックサイズを安易に無制限にする**のは**潜在的なリスクが大きい**

- 行儀の良い方法で対応するのであれば、**大容量バッファ**を受け渡す際は**8MB未満に分割**して送信するしかない

SGXの悪しき文化（1/4）

- この中で整数型 (uint16_t, uint32_t, uint64_t)をtypedefしたものはどれか？

```
sgx_isv_svn_t
sgx_prod_id_t
sgx_spinlock_t
sgx_misc_select_t
sgx_ra_context_t
sgx_enclave_id_t
sgx_time_t
```



全部

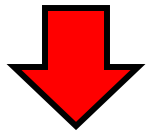
SGXの悪しき文化（2/4）

- シーリング (暗号化)を行う関数

```
sgx_seal_data(0, NULL, plain_len, plain,
    sealed_size, sealed_data);
```

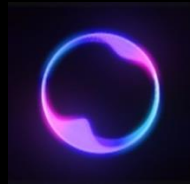
- アンシーリング (復号)を行う関数

```
sgx_unseal_data(sealed_data, NULL, 0, plain,
    plain_buf_size);
```



当たり前のように順序が逆

SGXの悪しき文化（3/4）

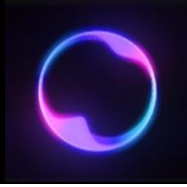


- 様々な機能制約 (Enclave内プログラム)
  - **システムコール**、一部**標準関数**、**動的ライブラリ**の禁止

- SGXSDKの煩雑な仕様
  - 使い勝手の悪い**独自のAPI・型**
  - ある論文「**412**行の非SGXプログラムを**SGX化**したら**3523行**になった」
  - **不親切**な**公式仕様書**、**Intel**の謎の**隠蔽体質**

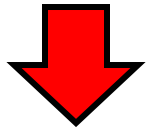
- 「境界管理」の概念
  - 自分が扱おうとしているデータが、**Enclave内外**のどちらにあるのか
  - 保護**境界をまたぐ関数**の**定義言語**の**複雑さ**

SGXの悪しき文化（4/4）



- EDLファイルにおける或る記述

```
public sgx_status_t store_vcf_contexts(sgx_ra_context_t context,
    [in, size=vctx_cipherlen]uint8_t *vctx_cipher,
    size_t vctx_cipherlen, [in, out, size=12]uint8_t *vctx_iv,
    [in, out, size=16]uint8_t *vctx_tag,
    [in, size=ivlen]uint8_t *iv_array, size_t ivlen,
    [in, size=taglen]uint8_t *tag_array, size_t taglen,
    [out, size=emsg_len]uint8_t *error_msg_cipher, size_t emsg_len,
    [out]size_t *emsg_cipher_len);
```



SGXは罪のない人間向けではない

SGXはコード安全性を保証しない

- SGX基本編で議論した通り、SGXは基本的にコードに対する保護や検証は一切行わない

- 何らかの理由でコードを守りたい場合、**SGXElide**[2]という研究成果や、Intel公式の**SGX-PCL**[3]は利用できる
  - しかし、前者はIASとはまた別の**第三者サーバを信頼**しなければならず、かつ**RAによるコード検証**が**事実上出来ない**
  - SGX-PCLは**RAも出来る**が、**Intelによるメンテナンスが放棄されている**

ARM TrustZoneにおけるケース

- **コード安全性の保証**が**対象外**であるのは、SGXだけでなく**TEE全般**に対して言える話である

- 例えば**サムスンのスマホ**は、スマホ内のTrustZone内のTEE内で動作する**Keymaster TA**において、**暗号鍵導出時**に**乱数的**でなければならない所を**決定的に行っていた**ため、攻撃により**暗号鍵を抽出されてしまう**という**脆弱性**を発見されている[12]

Intelの無責任さ（1/2）

- 基本的にIntelは無責任にも程がある
  - **RAのAPIのバージョンが上がる事**（v2→v3）で、IASリクエストの**旧バージョンとの互換性が喪失**[4]
  - 先程の**SGX-PCL**も、**sgx-ra-sample**も**管理を放棄**している
  - そのくせ**ライセンス**だけは**謎のものを使用**

- Linux-SGXでは、**モノトニックカウンタ**等を使用するための**PSEが突然廃止**されたため、Enclave外のデータとインデックスの対応付けをTrustedに行う事ができなくなった
  - あまりに急な出来事に、**コミュニティも狼狽**[5][6][7]
  - 後の攻撃編でも実践する「**シーリング再生攻撃**」の**対処が困難**となった

Intelの無責任さ（2/2）

- 挙げ句の果てには**第11世代以降CoreシリーズCPU**から**SGXを削除**
  - 今となっては**第3世代Xeonで継続**して搭載されているが、当時は**完全に削除**されると**専らの噂**となり、**大パニックが起きていた**

- いくら**製品や機能として優秀**であっても、**持続可能性に対する信頼性**を損なうと、**ユーザはその企業のサービスを敬遠する**
  - SDGsという単語はIntelにこそ送り付けてやるべき

SGX（TEE）の秘密計算利用の難しさ

RAの登場人物の奇妙なネーミング（1/2）

- ところで、RAでは**非SGX側**が**Service Provider**、**SGX側**が**Independent Software Vendor**と呼ばれていた

- しかし、特に**秘密計算的な文脈**では、SP（**サービス提供者**）の方が**SGX側**なのでは？という感覚がある

- また、ISV（**独立ソフトウェアベンダ**）というのも、**一体何を意味しているのか釈然としない**

RAの登場人物の奇妙なネーミング（2/2）

- SGXが本来想定していた利用モデルは、**秘密情報を有するサーバ**が、**SGX対応クライアント**に**Enclaveを配備**し、それを**RAで検証**してから**Enclaveに秘密情報を送る**というモデルであった
  - PowerDVDでのDRM処理の利用例がまさにこのケース

- つまり、**秘密情報を有する主体**（SP）自身が、**Enclaveイメージを生成してISVに配備**し、「**本当にISVは自分が作ったEnclaveを動かしているか？**」を**RAで検証**している
  - よって、SPは**予め控えておいた****MRENCLAVE**と**QUOTE内のそれ**を**比較**し、何の問題もなく検証する事が出来る

秘密計算モデルでのRAの致命的な欠点（1/3）

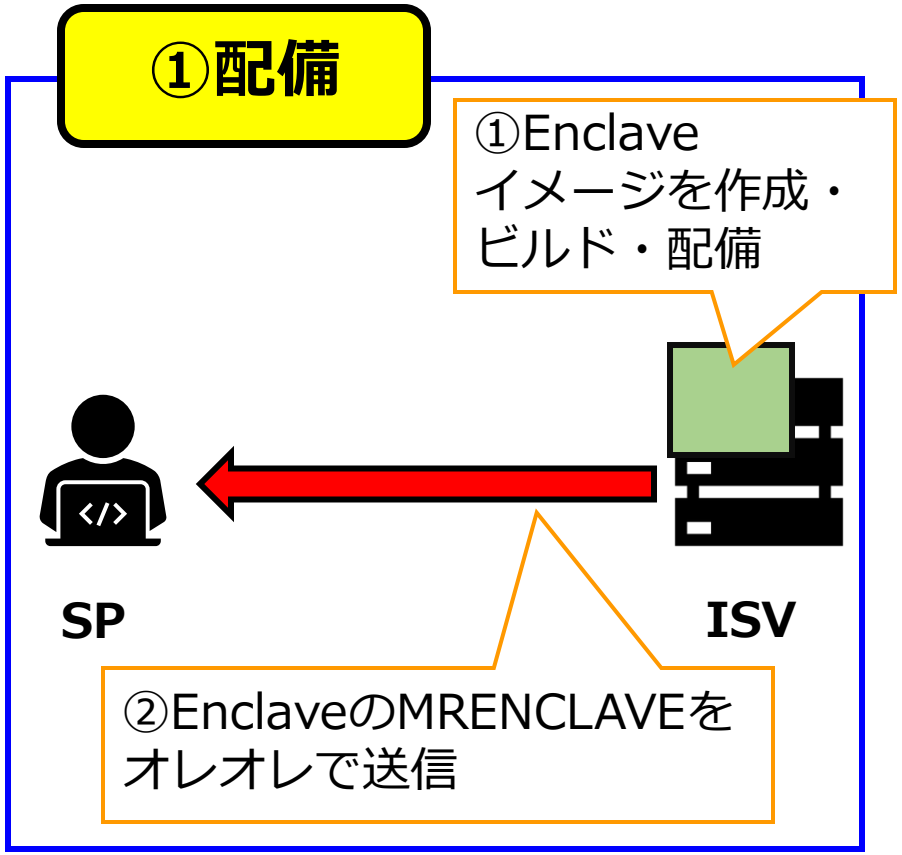
- しかし、サーバ側がSGX機能を提供する秘密計算モデルの場合、このRAは致命的な欠陥を抱えている

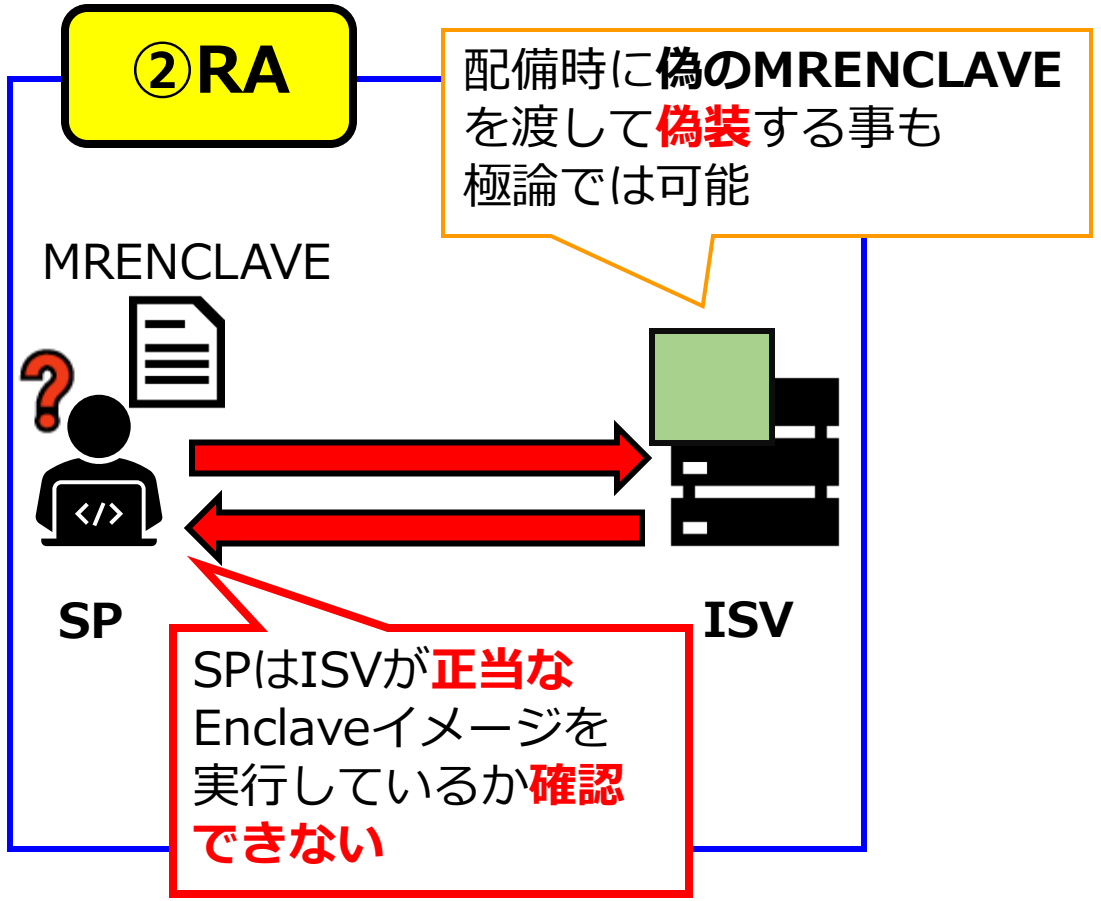
- SGXサーバ（**ISV**）側が**Enclaveコードを作成**し、それに基づくEnclaveを**ISV自身に配備**する場合、**SP**はどうやってそのEnclaveの**MRENCLAVEを検証**するのか？

- **作成・ビルド・配備全て**が**ISVで完結してしまっている**ため、例え**ISVからMRENCLAVEを受け取り**それに基づいてRAをしても、そのMRENCLAVEは**オレオレMRENCLAVE**でしかない

秘密計算モデルでのRAの致命的な欠点（2/3）

- つまり、秘密計算モデルでは例えRAを使っても、**相手のEnclaveの同一性**を**検証しようがない**という**致命的な欠点**を抱えている





秘密計算モデルでのRAの致命的な欠点（3/3）

- 一見、**SP側**でも**SGX環境を用意**し、そこでEnclaveイメージを**ビルド**して**MRENCLAVEを検証**するという手も考えられる

- しかし、MRENCLAVEは署名前**Enclaveイメージ**（=**コードだけでなくコンパイラ等にも依存**）や**SGXSDK**等にわずかでも差があれば**値が変わる**ため、ISVと**同一のものを導出出来ない事も多い**

RA問題の解決策を模索する（1/5）

■**SPがEnclaveを作成し、ISVに配備する**

- この方法であれば本来の利用モデルと合致し、RAで検証できる

- しかし、Azure等で自分で勝手にSGXアプリを作って動かす（**PaaS的運用**）ならともかく、クライアントにやらせている時点で**SaaSとしての運用が成立しない**

- そもそも、**この地獄のSGXプログラミングをユーザに強制するのですか？**
  - **そのような非人道的な真似は私が許さん**

RA問題の解決策を模索する（2/5）

■**運用で改善する**

- ユーザにEnclaveを開発させるのは非人道的なので、**ユーザにSGX環境を用意させ**、**Enclaveのビルドだけ**SPに行わせる
  - コードは**サービス提供側**（紛らわしいが**ISV**）が**提供**するか、**OSSから持ってくる**などをする

- これであれば、ユーザである**SP**は少なくとも**コーディングはしなくて良い**運用になる

- しかし、結局SPはその**コードの正当性**を**目視で確認**しなければ、**そのEnclave**を**信頼する事は出来ない**
  - **Enclaveコードを目視検証しなければならない時点で地獄**

RA問題の解決策を模索する（3/5）

■ISVでビルドされたEnclaveイメージを逆アセンブルする

- **正気か？**

- 例え署名済みEnclaveイメージでも**コード**は前述の通り**保護されない**ため、**逆アセンブルで動作定義を検証**する事は**理論上は可能**である

- 誰かしらに課す**拷問として利用**するのであればおすすめ

RA問題の解決策を模索する（4/5）

■**CAからコード一式のデジタル証明書をもらう**

- ソースコードやEnclaveイメージ、MRENCLAVE等の**一式**に対する**コードサイニング証明書**を**CA（認証局）**からもらう方法

- 公開鍵基盤という**脆弱な神話に甘える**方法なので、そもそもTEE的には**思想面**でも**好ましいとは言えない**

- さらに、このようなCAによる電子証明書は一般的に**年単位のスパン**である事が多いので、比較的**頻繁に更新の入る**Enclave周りの要素とは**絶望的に相性が悪い**

RA問題の解決策を模索する（5/5）

■**コンテナで検証する**

- ISVと**全く同じ環境**を**コンテナ**で**SPに提供**し、そこで**ビルド**して**MRENCLAVEを検証する**アプローチ

- **コード**は勿論、**コンパイラ**や**OS**レベルで**統一できる**ので、解決策候補の中では**最も現実的**

- コンテナが**そもそもSGXSDKを偽装していないか**等の**検証**は**別個必要**になる

TEEの秘密計算利用の難しさ

- このように、秘密計算モデルにおいてTEEをSaaS的に応用する場合、RAの根本的な設計に由来する困難がつきまとう

- これはAMD SEV等の**他のTEEにも共通する議論**であり、以下のように**何かしらの妥協の線引**を行わないと運用は難しい
  - ISVの提示するMRENCLAVEを全面的に信用する
  - 前述のようにSPがコードを受け取りビルドし、そのEnclaveイメージをISVに配備する、PaaSとSaaSの中間のような運用を行う
  - コンテナを検証するか無条件で信頼する前提で、コンテナによるMRENCLAVEの検証の手法を行う

SGX Fail

SGX Fail

- **SGXの運用**における、Intelの想定とユーザの実態、言い換えれば**理想と現実の乖離**に伴う**致命的な破綻**を糾弾している論文
  - 2022年に発表された、比較的新しい論文である

- 以下のような、**SGXの運用に伴うジレンマ**やそれに由来する**実世界での破綻例**を紹介している
  - マイクロコードアップデート（TCB Recovery）のタイムライン
  - Enclave開発者のジレンマ
  - **実世界での破綻例**（PowerDVD、SECRET Network）
  - IASに対するSigRLを悪用した潜在的なDoS攻撃

マイクロコードアップデートのTL（1/5）

- **TCB Recovery**には通常**マイクロコードアップデート**が伴うが、これは**BIOS（UEFI）アップデート**の形を取る
  - ちなみに、マイクロコードアップデートは恒久的に適用されるのではなく、マシンの起動の度にパッチを当てる方式になっている[8]

- BIOSアップデートはその**マザーボードのベンダ**から**配布される**ため、**ベンダが提供してくれない**と**ユーザ**は**マイクロコードアップデートによる修正**を**一切受けられない**

マイクロコードアップデートのTL（2/5）

- SGX Failは、6つのHWベンダについて、**Intel**により**マイクロコードアップデート**が**公開**されてから**何日でBIOSアップデートを提供**しているかを調査している
  - ASRock、Dell、HP、Lenovo、MSI、Gigabyte

- **SGXの脆弱性に対応**するような**BIOSアップデート**については、最短で**中央値25日**（HP）、最長で**中央値125日**（Lenovo）と、**リリース**までに**非常に時間がかかっている**事が判明した

マイクロコードアップデートのTL（3/5）

引用：SGX Fail[8]

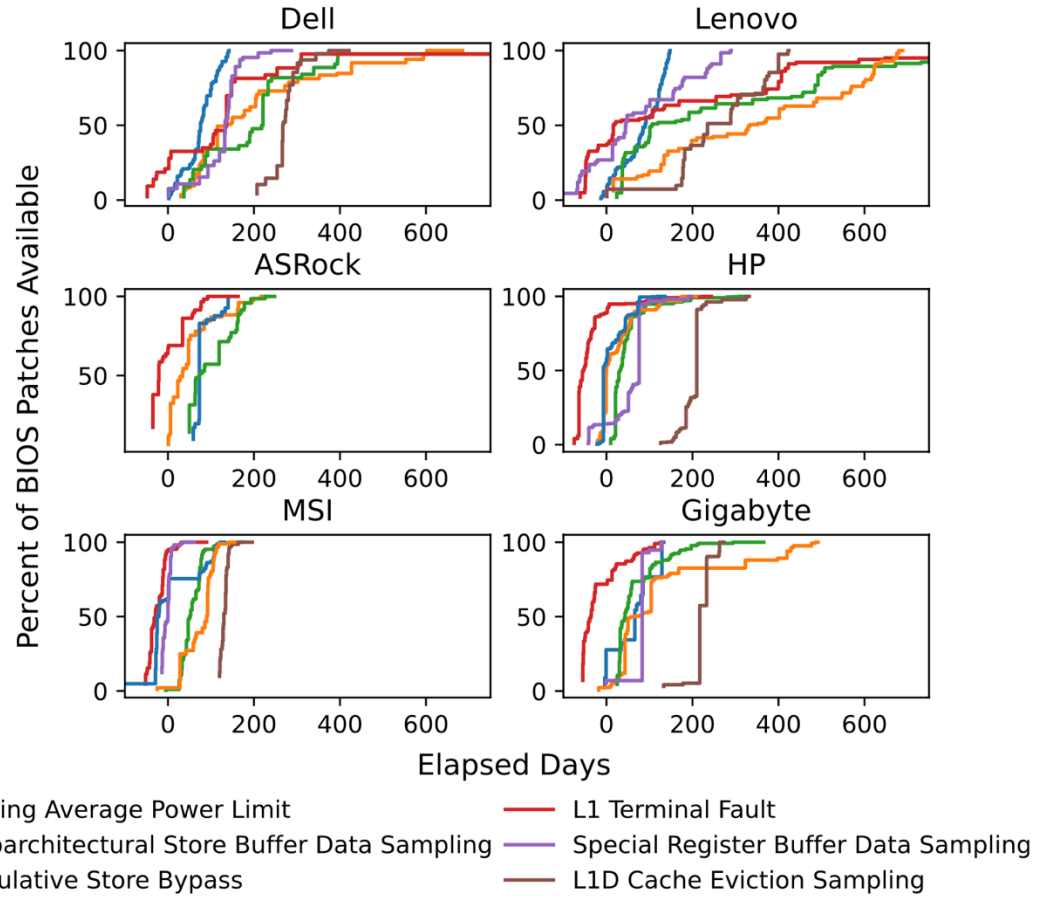


Figure 2: Time Taken To BIOS-Patch Major SGX Vulnerabilities After They Are Made Public

マイクロコードアップデートのTL（4/5）

- SGX向けでない**一般的なBIOSアップデート**の場合、最短で**中央値37日**（HP、MSI）、最長で**中央値329日**（Lenovo）となっている
  - AMDでのBIOSアップデートの場合、**Lenovoの中央値**は**1043日**

- そもそもBIOSアップデートはOSアップデートのように**頻繁に行う前提とはいいがたく**、しかも**マザーボードを損傷させるリスク**もあるため、SGX脆弱性対応のような**迅速な修正**との**相性**が**致命的に悪い**

マイクロコードアップデートのTL（5/5）

- さらに、Azure含む**クラウドベンダ**が提供しているSGX対応VMの場合、HWベンダから配布された**BIOSアップデート**を**クラウドベンダが適用**するまでにも**また時間がかかる**

Enclave開発者のジレンマ（1/3）

- 実は**このゼミのRAの実装**でも**既にこのジレンマに遭遇している**

- Humane-RAFW-MAAのデフォルトの実装では、
  SW_HARDENING_NEEDEDについては**看過するようにしている**

- このように、RAの受理条件を決める上で、**どこまで許すかにより発生するトレードオフ**に**Enclave開発者は苛まれる**

Enclave開発者のジレンマ（2/3）

- IASからの応答が**OKである場合のみ受理**するのが**理想ではある**が、**現実問題**としてそのように**丁寧に最新化しているマシン**は**少ない**
  - 先程の**BIOSアップデートの危険性や適用頻度**の議論も関連する
  - 例えば**9割のマシン**が**RA**で**弾かれて**しまっては、製品として**UXが最悪**となり、**お話にならない**

- かと言って、**条件を緩く**しすぎるとその分**膨大な量の脆弱性を看過**する事になり、結果として**SGXシステムの致命的な破綻に直結**してしまう
  - **PowerDVDの破綻例**がまさにこのケース

Enclave開発者のジレンマ（3/3）

- マイクロコードアップデートに限らず、例えば**SGXSDK**についても、**平然と互換性を喪失させる**ようなアップデートをねじ込んでくる**Intelの無責任さ**も看過できない
  - **PSE**の**突然の削除**による**モノトニックカウンタ**や**信頼可能時間ソース**の**剥奪**が**顕著な例**

SECRET Networkの破綻例（1/9）

- **SECRET Network**：スマートコントラクトの実行をSGXの**Enclave内で行い**、**トランザクションも暗号化**する事を特長としている**ブロックチェーン**の1つ
  - **スマートコントラクト**：**ブロックチェーン**上での**処理・取引**を行うに当たって**自動実行されるプログラム**。例えばスマートコントラクトによる**検証の結果**、**特定の条件**を満たし**取引が許可された場合のみ**実際に**自動的に取引が行われる**、等の運用が出来る
  - **トランザクション**：そのブロックチェーン上における**取引履歴**の事。ブロックの中身そのものでもある

画像出典：https://learn.bybit.com/defi/what-is-secret-network/

SECRET Networkの破綻例（2/9）

- 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**暗号化**し、その**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

SECRET Networkの破綻例（3/9）

- 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

SECRET Networkの破綻例（4/9）

- 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**

SECRET Networkの破綻例（5/9）

- 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**

SECRET Networkの破綻例（6/9）

- ノード初期化関数でコンセンサスシードを復号し初期化する際に発生する、IPFSL鍵の**AES鍵伸長処理**に対し、**ÆPIC Leak攻撃**を仕掛けた
  - ÆPIC Leakの詳細は攻撃編セクションで解説
  - 鍵伸長処理への到達の特定には、これまた攻撃編で解説する**Controlled-Channel Attacks**で提案された手法を用いている

- ÆPIC Leakにより得られた**IPFSL鍵の断片情報**から、一般的かつ容易な手法で**IPFSL鍵を完全に復元**し、**コンセンサスシードを復号し取得**する事に成功している
  - 断片情報からの復元方法についてはここでは割愛。SGX特有の要素は一切存在しない一般的なものである

SECRET Networkの破綻例（7/9）

- コンセンサスシードが漏洩してしまうと、所有する個人だけで閲覧し楽しむNFTである**プライベートNFT**まで**暴露出来てしまう**
  - 画像はSGX Failより引用

Figure 5: Example NFT that we created and then decrypted.

SECRET Networkの破綻例（8/9）

- 同様の方法により、ÆPIC Leakによってそのノードの**EPID Attestationキーの抽出**にも**成功している**
- SGX Failの論文執筆時点で、前述した**唯一の回復策**である**ハードフォークの実施**をSECRETは計画している
- **応急措置**として**危険なノード**からの**コンセンサスシードの削除**等が行われている
- しかし、**ゼロデイ的**かつ**ステルス**に**世界の誰かが既にこの攻撃を仕掛けたかも知れない**という、非常に**疑心暗鬼的かつ苦しい状況**に**SECRETを陥れる結果**となっている

SECRET Networkの破綻例（9/9）

- SECRETの事例は、**Intelが脆弱性を開示**してから、実際に**マイクロコードアップデートを頒布**するまでに**ラグがある**という**運用上の問題**を**浮き彫りにしている**
  - ÆPIC Leak脆弱性開示が2022/8で、TCB Recoveryは2022/11末
  - 元々はTCB Recoveryは2023/3予定であったため、SGX Failの報告を受けて早回しにした形となっている
  - 仮にTCB Recoveryが展開されても、前述の**ベンダ**による**展開速度の問題**や**ユーザがそれを適用するかという問題**が依然つきまとう

PowerDVDの破綻例（1/7）

- **PowerDVD**：Cyberlinkが提供している動画再生PCソフト。
  Ultra HD Blu-ray Disk（**UHD-BD**）の再生時に、**SGXによって**
  デジタル著作権管理を行っている事で知られている

- UHD-BDディスクの内容は**AACS2鍵**という**専用のAES鍵**で
  暗号化されており、**AACS2**というアルゴリズムに従って
  内容を復号し再生する
  - AACS2は**ディスクの内容を暗号化**するための**コピーガード**であって、
    再生処理中の**メモリ上の映像データを保護するものではない**点に注意

PowerDVDの破綻例（2/7）

- UHD-BDを再生するSGXマシンは、PowerDVDの提供する**CLKDE**（CyberLink Key Downloader Enclave）を起動し、Cyberlinkの**AACS2鍵プロビジョニングサーバ**により、**CLKDEを検証するRA**が**実行**される
  - 鍵を提供する**サーバがSP**、再生する**クライアントがISV**である、SGXの元々の利用モデルに見事に合致している運用例である
- **RAに成功**すると、Cyberlinkのプロビジョニングサーバは**AACS2鍵**を**CLKDEにデプロイ**し、CLKDEは**MRSIGNERポリシ**でAACS2鍵を**シーリングし保存**する
- UHD-BD再生時は、この**AACS2鍵で内容を復号し再生**する

PowerDVDの破綻例（3/7）

- 逆に言えば、AACS2鍵は**この世のあらゆるUHD-BD**を**復号できる鍵**であるため、これが漏洩するとどのマシンだろうが**全てのUHD-BDの内容を復号**出来てしまう
  - これに対応するにはその鍵を**失効**させなければならないが、**既存のUHD-BD**のセキュリティを**回復させる事はできず**、漏洩した鍵の**失効後に製造されるUHD-BD**で**セキュリティ保証を回復させる**事しかできない

- にも関わらず、PowerDVDは**IASからのRA応答**でも**一番許してはいけない**類のステータスである**GROUP_OUT_OF_DATE**をも**許容する実装**となっている

PowerDVDの破綻例（4/7）

- SGX Failでは、GROUP_OUT_OF_DATEであるようなマシンに対し**Foreshadow攻撃**を仕掛け**EPID Attestationキーを抽出**した
  - Foreshadowの詳細は攻撃編で解説

- 抽出したAttestationキーを用いて、**REPORTの検証をスキップ**し**任意のMRENCLAVE/MRSIGNER**で**差し替えた偽造REPORT**に対し**EPID署名**する、**偽造QE**を実装している

- これを中心に、SGXの**Enclaveの動作**を**全てEnclave外**で**エミュレート**する、Emulated Guard Extentions（**EGX**）というとんでもない**SGXエミュレータ**まで実装している

PowerDVDの破綻例（5/7）

- EGXにおける**偽造QE**で**偽造されたQuote**は、Cyberlinkによる**正規のMRENCLAVE/MRSIGNER**を含むように**改竄している**ため、Cyberlink**のAACS2プロビジョニングサーバ**はそれが**正当**であると**誤認して****AACS2鍵を送信**する

- しかし、**EGX**は実際には**Enclave外で動作**するため、**全く保護されていないメモリ上**に**AACS2鍵が提供されてしまう**事になる
  - **AACS2鍵**が手に入れば、**SGX**どころか**Intel CPUですらないマシン**でも**UHD-BDの内容**を**任意に復号**出来てしまう

- そして、本来**公開されてはならないAACS2アルゴリズム自体**も論文中の攻撃により**解読され暴露**されてしまっている

PowerDVDの破綻例（6/7）

- 言うまでもなく、**GROUP_OUT_OF_DATEを許容**していた事が原因
  - RAから返ってくる応答の中でも**最も許可してはならない**類の応答

- しかし、一般的に製品の**ユーザの殆ど**は**非技術者層**であり、**安全性**よりも**有用性を圧倒的に重視**する
  - 何なら、ユーザからしてみれば**コピーガード**含む**DRM**はユーザからしたら**邪魔**でしか無い

PowerDVDの破綻例（7/7）

- **有用性が落ちる**とユーザは**PowerDVD**を**購入してくれなくなる**ので、**非常に条件の厳しいOK応答だけを通す**設定にするのは、**商業的には論外**である

- このように、**安全性と有用性のトレードオフ**の末に、本来のSGXの安全性を損なってしまう（how **S**tuff **G**ets e**X**posed）極めて**苦しい選択**を、**大衆向け商用展開**では迫られてしまう
  - その意味では、前述の通り本来SGXの想定していない**秘密計算モデル**の方が、こういったトレードオフ問題は**サービス提供側**（サーバ）で**対処できる**ので解決しやすい

SGXキー漏洩時の対応

SGXキー漏洩時の対応（1/3）

- **SGXの安全性**は、**Enclave内で導出**される**様々な鍵**が**漏洩しない前提**のもとに成り立っている

- しかし、2025年現在となっては数多くの攻撃により、例えば**レポートキー**や**EPID Attestationキー**が**抽出**されている

SGXキー漏洩時の対応（2/3）

- レポートキーが漏洩すると、**SECS**によって**保証されていない**
  任意のMRENCLAVE等に対する**REPORT**を**Enclave外で**
  偽造出来るようになってしまう（**EREPORTの迂回**）
  - **LA**及び**RA**の**検証者を騙す**のに悪用可能

- Attestationキーが漏洩すると、**任意のREPORT**に対して**署名**して
  偽造QUOTEを生成し、あたかも正当なEnclaveであるかのように
  VerifierやRPを騙す事が出来てしまう（**QEの迂回**）
  - Verifierは**気付かずにQUOTEは改竄されていないという応答を返してしまう**
  - **RA**の**検証者を騙す**のに悪用可能

SGXキー漏洩時の対応（3/3）

- これらは、**TCB Recovery**を行う事で**いずれも対処できる**
  - レポートキーは**CPUSVN（マイクロコードバージョン等を材料に導出）**に依存して生成される[9]ため、TCB Recoveryで**マイクロコード更新**を行えば**鍵が変わり偽造不可**となる
  - TCB Recoveryの際は、**再プロビジョニング**によりAttestationキーを更新するため、QUOTEも**偽造不可**となる

- では、**上記の鍵が漏洩**しているにも関わらず「**TCB Recoveryを実施しない**」という選択を取る**悪性のSGXマシン**を**どのように識別**しているのか？
  - ここではあくまでも**RAの場合**について議論する（LAは考慮しない）

レポートキーのみ漏洩した場合

- **レポートキーのみ**の漏洩の場合、そのマシンの**PCK Cert**を**PCK CRL**に登録してもらう事で対処可能
  - これらの鍵漏洩時のどのシナリオにも共通するが、基本的に**Intelに頼んで失効リストに登録してもらう**しか無い

- ちなみに、EPID-RA時代には、漏洩したAttestationキーに紐づく署名を失効させる仕組みが存在した（SigRL）

Attestationキーのみ漏洩した場合

- こちらの場合も、Intelに依頼し**そのマシンのPCK Cert**を**PCK CRL**に登録させる事で対処可能

- **PCKCRL**と**PCK Cert**を比較し、**一致したら失効済み**なので、VerifierはRA応答として**SGX_QL_QV_KEY_REVOKED**をRPに返却する[10]

- こうなるとAttesterは**TCB Recoveryを行う**以外に**RPを信頼**させる**手段**が**存在しなくなる**

レポートキーとAttestationキー双方が漏洩した場合

- この場合も**PCK CertをPCK CRLに登録**する事で対処可能である

- この場合レポートキーは、**Attestationキー**を**再プロビジョニング**すると**TCB Recovery**で**マイクロコードも更新**されている事になるので、**CPUSVNの変更**により**漏洩したものは使用不可**になる

本セクションのまとめ

- 様々な観点及びレイヤから、SGXやTEEの抱える数々の難しさについてある程度詳細に議論した

- このセクションの内容は直接コーディングスキルに紐づくものではないが、SGXやTEEを用いたシステムのセキュリティ設計を行う上では極めて重要なものとなっている

参考文献

[1]”Segfault While passing a big data through an Ocall #376”, GithHub, https://github.com/intel/linux-sgx/issues/376

[2]“SgxElide: Enabling Enclave Code Secrecy via Self-Modification” by Erick Bauman et al., https://web.cse.ohio-state.edu/~lin.3021/file/CGO18.pdf

[3]”linux-sgx-pcl”, GitHub, https://github.com/intel/linux-sgx-pcl

[4]”Error 400 from IAS for Quote Attestation Request #27”, GitHub, https://github.com/intel/sgx-ra-sample/issues/27

[5]”SealedData example missing Platform Service capability”, GitHub, https://github.com/intel/linux-sgx/issues/541

[6]”The delay attack towards the trusted time”, Intel, https://community.intel.com/t5/Intel-Software-Guard-Extensions/The-delay-attack-towards-the-trusted-time/m-p/1343497#M5056

[7]”No longer compatible with SGXSDK v2.8 or newer version #48”, GitHub, https://github.com/intel/sgx-ra-sample/issues/48

参考文献

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

[9]”SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution”, Guoxing Chen et al., https://arxiv.org/pdf/1802.09085.pdf

[10]”Attestation Service for Intel® Software Guard Extensions (Intel® SGX): API Documentation”, Intel, https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf

[11]”SGAxe: How SGX Fails in Practice”, Stephan van Schaik et al., https://sgaxe.com/files/SGAxe.pdf

[12]”Trust Dies in Darkness: Shedding Light on Samsung’s TrustZone Keymaster Design”, Alon Shakevsky et al., https://eprint.iacr.org/2022/208.pdf