8. SGX攻撃編①

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に対する攻撃はおびただしい数存在するが、その内のいくつかをタイプ別に分類して紹介する

- 一部の攻撃に対する防御策をSGX-Vaultの実装に組み込む

SGXに対する数多くの攻撃

SGX攻撃の分類（1/2）

- SGXの分野では、SGXを利用する研究も活発である一方で、同じくらいにSGXに対する攻撃に関する研究も多い

- 前セクションの主題であった**SGX Fail**の論文や、**ÆPIC Leak**という攻撃の論文等でも、**SGXへの攻撃（SGXの脆弱性）の分類**を行っている

- 今回は、キャンプで紹介しきれる範囲で攻撃をピックアップし、**SGX101**における攻撃の分類をベースに分類して紹介する

SGX攻撃の分類（2/2）

- 本ゼミでは以下のように攻撃の分類を行う：

攻撃分類	攻撃例	主な責任の所在
メモリ破壊攻撃	Dark-ROP	Enclave開発者
ページフォールトベースの サイドチャネル攻撃	Controlled-Channel Attacks	Enclave開発者
物理現象を悪用する サイドチャネル攻撃	SGX-Bomb、Plundervolt	Intel、ハードウェアベンダ
再生攻撃	シーリング再生攻撃	Enclave開発者
マイクロアーキテクチャ（μ-Arch） に対するサイドチャネル攻撃	Branch Shadowing Attack	Intel
過渡的実行攻撃 - Spectre型	SgxPectre、ITS	Intel
過渡的実行攻撃 - Meltdown型	Foreshadow、MDS、GDS	Intel
過渡的実行攻撃 - 融合型	LVI、GVI	Intel
初期化不良	ÆPIC Leak、SGX-Bleed	Intel

メモリ破壊攻撃

- Enclaveの外からメモリ破壊を用いて攻撃は出来ないと基礎編で説明したが、**Enclave内**から実行できる場合は別

- 代表例としてDark-ROP[4]が挙げられる

Dark-ROP（1/2）

- 名前の通り、ベースは**ROP攻撃**
  - バッファオーバーフロー等によりリターンアドレスを破壊し、攻撃対象コード内に潜む「ガジェット」と呼ばれる攻撃コードを用いて、攻撃者による任意の攻撃を行う攻撃

- Enclaveの**中身は直接見えない**ため、ファジングによるメモリ破壊に伴うページフォールト等の周辺情報から、**サイドチャネル的**に**脆弱箇所**や**攻撃ガジェット**の特定を行う

Dark-ROP（2/2）

- バッファオーバーフロー等を起こせるような**バグ**が**コードに存在していなければ**Dark-ROP攻撃は**無力**である

- このようなコード脆弱性を**コンパイルの段階で拒絶**してくれる**Rust**を用いてSGX開発できる、Apache Teaclaveの**Rust-SGXSDK**を用いるのは**かなり効果的**

ページフォールトベースのサイドチャネル攻撃

- **ページフォールトの有無**や、**OSに通知**される**ページフォールトアドレス**等をヒントとしながら秘密情報を推測するサイドチャネル攻撃

- 圧倒的に**Controlled-Channel Attacks**[6]が**有名**であり、**他の様々な攻撃**においても**攻撃の一部分**として**用いられている**

- Controlled-Channel Attacksについては後の**詳細編セクション**で詳述

物理現象を悪用するサイドチャネル攻撃

- 文字通り、**物理的な現象**を**悪用・観測**する事によって、**秘密情報の推測や抽出**、あるいは**SGXの妨害**を行う類の攻撃

- 例としては**SGX-Bomb**[7]、Plundervolt[8]、PLATYPUS[9]が挙げられる
  - Plundervoltは、モデル固有レジスタというインタフェースを通じてCPUの動作電圧を一瞬下げる事で、故障注入を実現する攻撃。
    その後差分故障解析等の攻撃に繋げて秘密情報を漏洩させる
  - PLATYPUSは、RAPLという消費電力変動モニタリングインタフェースを通じて電圧を測定し、その電圧からEnclave内の秘密情報を推測するサイドチャネル攻撃

SGX-Bomb

- **メモリの特定の箇所**に**アクセスを集中**させ、**過剰な電荷を発生**させて周辺の**メモリセルのビットを反転**させる**RowHammer攻撃**のSGX版

- ところで、SGXにおいては、**EPCの内容に改竄**が発生すると**MEE**が**自動的にマシンをシャットダウン**する

- この**MEEの仕様**を**悪用**し、**SGX-Bomb**によってSGXを利用するマシンに対する**サービス妨害（DoS）攻撃**を行えてしまう
  - わざとエラーを発生させ、マシンの電源を落とさせて処理を妨害する

再生攻撃（Replay Attack）

- 傍受した**暗号文**をそのままシステムに**再度送信する**（リプレイ）事により、**暗号文自体以外の情報**を**知らずとも**、その**システムの何らかの機能**を**利用できてしまう**攻撃
  - 一般的なシナリオでは、**認証データの暗号文の再送**が挙げられる

- SGXでは、**シーリング再生攻撃**が最も**有名かつ致命的**である
  - **本セクションの後半で詳述**

- 参考文献[10][11]によれば、**MEEではなくTME-MK**を使用している、**第3世代Xeon-SP**上でのSGXでは**完全性保証が失われている**ため、**EPCの再生攻撃**も**実現してしまう**可能性がある

μ-Archに対するサイドチャネル攻撃

- **μ-Arch：命令セットアーキテクチャ**よりも**ローレベル**な、**CPUの内部構造やデータフロー**を**定義**する**設計レベル**の事
  - 有名所としては**キャッシュメモリ**もμ-Archに含まれる
  - その他、**直近の分岐履歴**等を記録する**LBR**（Last Branch Record）や、アウトオブオーダー実行等で未処理のストア命令を記録しておく**ストアバッファ**等が存在する

- μ-Archに対する攻撃は、**過渡的実行攻撃**（Transient Execution Attacks）の**アウトブレイクが発生**した**2018年**以降**急激に増えている**
  - この分類では、**過渡的実行に依存しない類の攻撃の例**のみを挙げ、**過渡的実行攻撃**に関しては**別分類**としている

μ-Archに対するサイドチャネル攻撃

- この分類としてはまず、**従来型**の**キャッシュサイドチャネル攻撃**を**SGX**に対しても**実行可能にしたもの**[12]がある

- また、**SGX**が**Enclave脱出時**に**LBRの分岐履歴**を**消去しない**という仕様を悪用し、履歴からデータがどのように使用されたかを推測して秘密情報を推測する、**Branch Shadowing Attack**もこれに含まれる

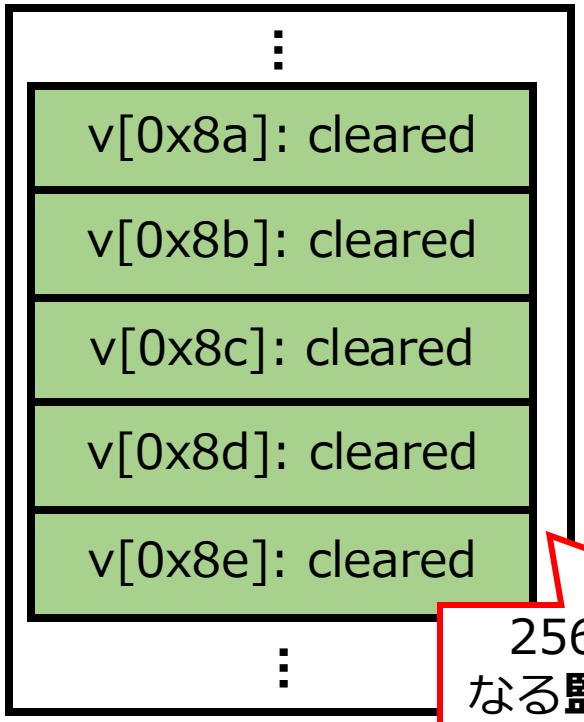
キャッシュサイドチャネル攻撃の種類（1/3）

- **キャッシュサイドチャネル攻撃**：キャッシュアクセス時に発生する**キャッシュヒット/ミス**に起因する**アクセス時間の違い**から、**キャッシュの使用状況を推測**し、そこから**使用された秘密情報を特定**するタイミング攻撃

- SGXへの攻撃では、**FLUSH+RELOAD**型攻撃と、**PRIME+PROBE**型攻撃の2つが特に使われる事が多い

キャッシュサイドチャネル攻撃の種類（2/3）

- **FLUSH+RELOAD**：キャッシュラインを**フラッシュ**（クリア）し、攻撃対象に何らかの活動させた後、**再度アクセス**する攻撃[14]
  - **再アクセス時**に**アクセス時間が短ければ**、**そのキャッシュ値**を**攻撃対象が使用**したという事がわかる（データ自体の推測）



キャッシュ

256要素（=1バイト）からなる**監視用配列**の各要素に対応する**キャッシュをフラッシュ**

秘密情報0x8cを**インデックス**として**監視用配列にアクセス**

```
ret = func();
```

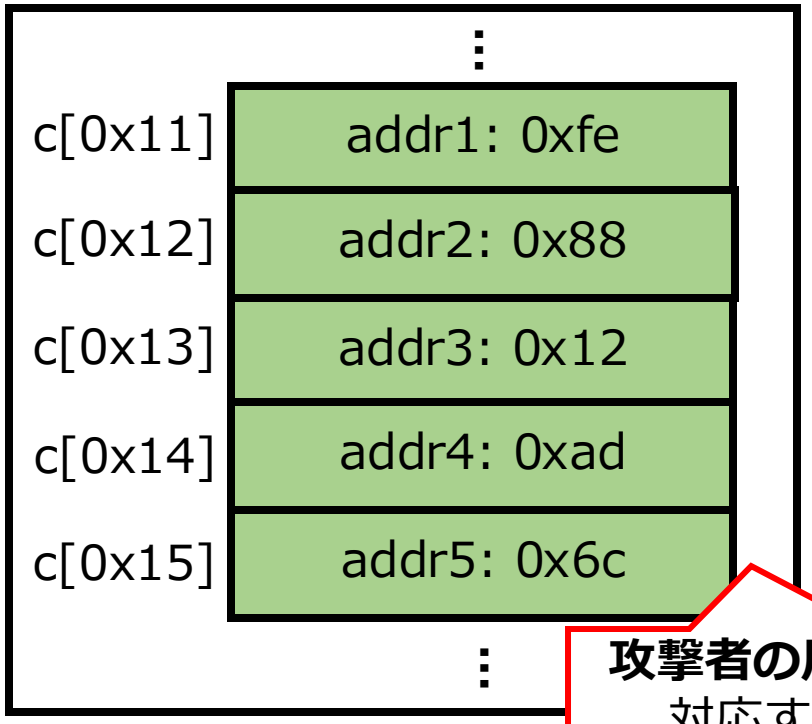
攻撃対象コード

監視用配列インデックス	アクセス時間[ナノ秒]
0x8a	1.52
0x8b	1.76
0x8c	**0.80**
0x8d	1.47
0x8e	1.63

監視用配列の全スロットに**アクセス**し、**アクセスが速かったもの**はキャッシュにあり**funcに使用された**（=**秘密バイトが0x8c**）と分かる

キャッシュサイドチャネル攻撃の種類（3/3）

- **PRIME+PROBE**：キャッシュを**攻撃者のデータで埋め尽くし**、攻撃対象に何らかの活動をさせ、**再度アクセス**する攻撃[14]
  - **再アクセス時**に**アクセス時間が長ければ**、**その場所のキャッシュ**を攻撃対象が使用したと分かる（場所の推測）



キャッシュ

攻撃者の用意したaddr1～5と対応する値でキャッシュc[0x11]～c[0x15]を**埋める**

何らかの処理を行い、**攻撃対象**が**キャッシュc[0x13]**を使用

```
ret = func();
```

攻撃対象コード

キャッシュ	アクセス時間[ナノ秒]
c[0x11]	0.82
c[0x12]	0.93
c[0x13]	**1.75**
c[0x14]	0.90
c[0x15]	0.87

対象アドレス全てに**アクセス**し、**アクセスが遅かったキャッシュ**が**funcに使用された**（=**場所c[0x13]を使用した**）と分かる

アウトオブオーダー実行と投機的実行

- **アウトオブオーダー実行**：プログラムに記述された**命令順に関係なく**、準備のできた（処理に必要なデータが揃った）命令から実行する手法

- **投機的実行**：近い将来に使われるかも知れない処理を、必要であると確定する前に**先行して実行**し**結果を用意**しておく手法

過渡的実行攻撃

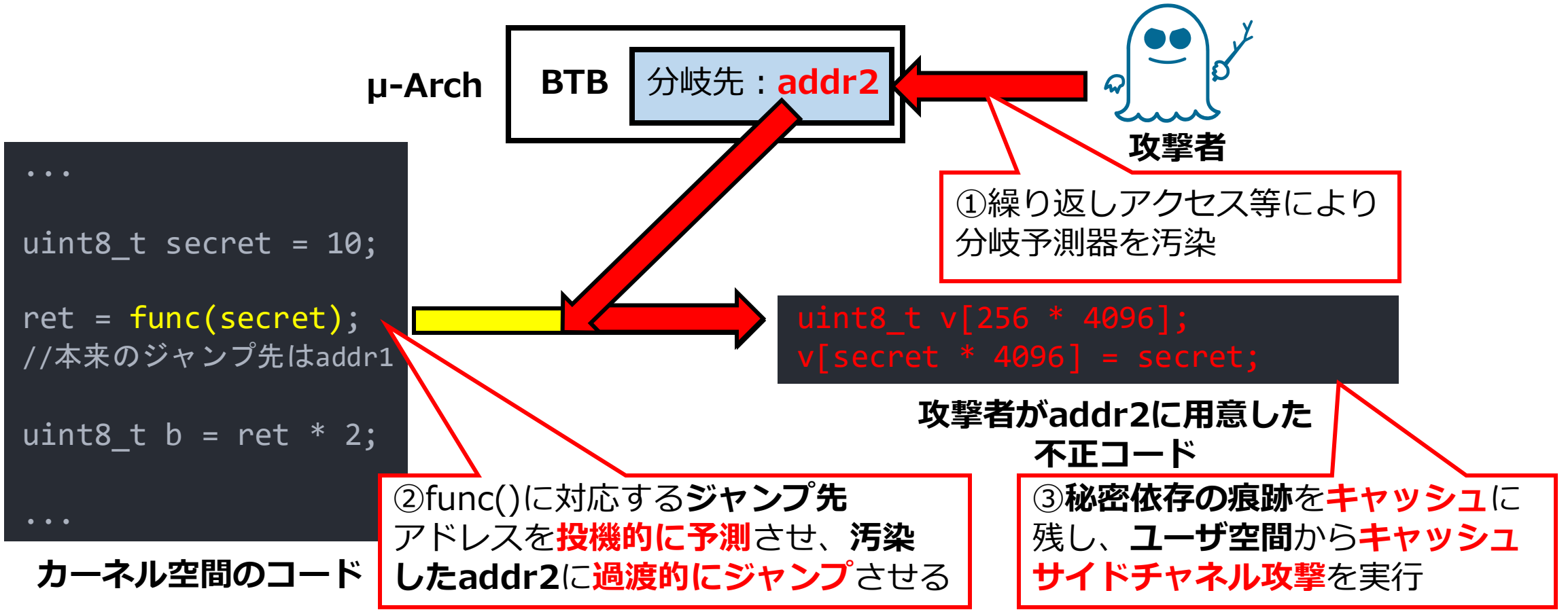
- **過渡的実行攻撃**：**投機的実行**において、**本来使われてはいけない秘密情報**が使用されてしまい、**投機的実行結果**として残された**秘密情報に依存する痕跡**から**秘密情報を抽出**する攻撃
  - **Transient Execution Attacks**の訳

- このような投機的実行の結果は、本処理が投機的実行に追いついて（=**命令リタイア**）**投機失敗と判断**すると共に**破棄される**が、**破棄前**に**キャッシュ等**に**痕跡を残してしまう**のがこの攻撃の要

- 非SGX環境でも有名な攻撃例に、**Spectre**攻撃と**Meltdown**攻撃が存在する

Spectre攻撃

- 分岐予測を行うμ-Arch上のバッファ（分岐対象バッファ；BTB等）を、**不正なコードを配置**した**アドレス**で汚染した状態で投機的実行を誘発させる事で、不正コードを投機的に実行させる攻撃



Meltdown攻撃

- ある命令で**例外を発生**させる事などにより**投機的実行を誘発**させ、**後続の秘密依存**の**過渡的実行**により**キャッシュ**等に**痕跡を残し**、**秘密情報を抽出**する攻撃
  - Spectreと違い、攻撃コードは攻撃者が自前で用意し実行できる

攻撃者

①**本来アクセスできない秘密情報**にアクセスを試みる攻撃コードで、**フォールト**等により**例外を発生**させ**投機的実行を誘発**させる

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

攻撃者の用意したコード

②**投機的実行**により**秘密依存の処理が実行され**、**秘密依存の痕跡**を**キャッシュ**等に残して**秘密を抽出**する

過渡的実行攻撃 - Spectre型（1/2）

- SGXに対するSpectre型の過渡的実行攻撃としては、**SgxPectre**攻撃が挙げられる

- 文字通り、**Enclave内のコード**においてSpectre的な**分岐予測汚染**による**不正コードへの過渡的な遷移**を発生させ、**FLUSH+RELOAD**キャッシュサイドチャネル攻撃で**秘密情報を抽出**する攻撃

過渡的実行攻撃 - Spectre型（2/2）

- また、2025年には、**Indirect Target Selection（ITS）**というSpectre-v2タイプの脆弱性も発見された[18][19]

- 従来のSpectre-v2では異なる特権モードから汚染をしていた（例：ユーザ分岐でカーネル分岐を汚染）所を、**同一特権モード内で汚染し攻撃**している
  - 特定の条件において、攻撃用直接分岐で攻撃対象間接分岐をSpectre的にハイジャックする
  - Spectre対策は専ら**ドメイン分離ベースが多い**ため、**同一特権内完結**の**ITS**には**そのような対策は無力**である

過渡的実行攻撃 - Meltdown型

- SGXに対するMeltdown型の過渡的実行攻撃としては、
  Foreshadow（L1 Terminal Fault ; L1TF）攻撃が挙げられる

- **Foreshadow**については**後のセクション**で解説

過渡的実行攻撃 - 融合型

- **Meltdownにより漏洩**する情報を**Spectre的な注入**に繋げる事で、広範な攻撃対象において秘密情報の漏洩を可能とする過渡的実行攻撃

- **Load Value Injection（LVI）**、**Gather Value Injection（GVI）**と呼ばれる攻撃がこの分類に含まれる
  - SGXに対する攻撃の中でも**攻撃難易度が最難関**といっても過言ではない攻撃

- **LVI**については**後のセクション**で解説

初期化不良

- **メモリ**や**アーキテクチャ上のバッファ**等を、**十分に初期化していない**事によって発生する攻撃

- この分類に含まれるSGXに対する攻撃としては、**SGX-Bleed**と**ÆPIC Leak**が例として存在する
  - SGX-Bleedは**本セクションで後ほど実践**
  - ÆPIC Leakは**後のセクションで解説**

SGX-Bleed実践

SGX-Bleed（1/5）

- **構造体**における**パディング**部分が**正しく初期化されていない**事を悪用し、**未初期化部分**の**秘密情報**を**Enclaveから漏洩させる攻撃**

- Enclave開発者による**不十分な実装が主な原因**であるため、開発者が注意する事により**比較的容易に対策可能**

SGX-Bleed（2/5）

- 以下のような構造体を考える：

```
typedef struct {
    uint64_t a;
    uint8_t b;
    uint64_t c;
} test_struct_t;
```

- **直感的**には、この構造体のサイズは8+1+8で**17バイト**であるように見えるが、実際にsizeof(test_struct_t)で**サイズを取得**すると、恐らく**24バイト**になっている

SGX-Bleed（3/5）

- これは、**メモリアクセスの効率化**等の観点から、真ん中の**uint8_t型メンバ**の後に、**uint64_t型のサイズ**に合わせるように、**7バイトのパディングが挿入**されるためである
  - 8+1+7+8=24であるため、sizeofによる結果に一致する

```
typedef struct {
    uint64_t a; //8bytes
    uint8_t b; //1byte
    //7bytes padding
    uint64_t c; //8bytes
} test_struct_t;
```

SGX-Bleed（4/5）

- この構造体をECALLから**戻り値としてリターン**する時、**Edger8r**は以下のように**構造体を丸ごとコピー**するため、**パディング部分ごと** **Enclave外にリターン**する事になる

```
sgx_status_t ecall_bleed(sgx_enclave_id_t eid, test_struct_t* retval)
{
        sgx_status_t status;
        ms_ecall_bleed_t ms;
        status = sgx_ecall(eid, 0, &ocall_table_Enclave, &ms);
        if (status == SGX_SUCCESS && retval) *retval = ms.ms_retval;
        return status;
}
```

- さらに、構造体の**各メンバに代入**しても、**パディング部分**に対して**内容の更新がなされる事はない**

SGX-Bleed（5/5）

- もし**この構造体を生成する前**に、**同じ場所**で**過去に秘密情報を含むバッファが展開**されており、**内容がクリアされないまま解放**されていた場合、**パディング部分**を通じて**秘密情報がEnclave外に漏洩**してしまう
  - Use-After-Free脆弱性の一種であるとも言える
  - **去年の応募課題Q3-2がまさにこの脆弱性を問う問題**

- 対策としては、
  - 構造体作成時に**パディング込みのサイズ**に対して**memsetでクリア**する
  - #pragma packで**パディングを無効化**する
  - バッファ解放前には必ず**内容をmemsetやcalloc等でクリア**しておく

  等が挙げられる

SGX-Bleed実践（1/2）

■**SGX-Bleed実践課題**
Enclave内において、**’a’**で埋められた**uint8_t型の秘密バッファ**を用意する（サイズは**24バイト以上推奨**）。その後、内容を**初期化しないままfree**し、前述の**構造体test_struct_tを宣言**する。

このtest_struct_tの**パディング部分**を通じて、未初期化の秘密バッファから**7バイトの’a’の羅列**を、**SGX-Bleed**により**Enclave外に漏洩**させよ。

SGX-Bleed実践（2/2）

■SGX-Bleed実践課題の要件・ヒント

- 必ずしも解放した秘密バッファと同じ場所に構造体が展開されるとは限らないため、秘密バッファのアドレスをuint64_t型で控えておき、必ず同じ場所に展開させるようにする

- Enclave外にリターンさせた構造体の内容の各バイトの確認には、OpenSSLライブラリのBIO_dump_fp関数が便利

シーリング再生攻撃の実践と対策

シーリング再生攻撃（1/2）

- シーリングしたデータを**ファイルとして保持**しておくというユースケースは、**SGXにおいてごく一般的**である

- しかし、**ファイル名**に基づくシーリングデータの**ファイル読み込み**は**OS**、つまり**信頼不可能領域での処理**である

- ところで、シーリングではポリシに応じて、**同一のMRENCLAVE**または**MRSIGNER**のEnclaveでシーリングされたデータは、その同一性情報が**同じであるようなEnclave**で**アンシーリング**出来る

シーリング再生攻撃（2/2）

- SGX-Vaultを例に取ると、例えば**別のユーザ**が**シーリング**したデータであっても、**ファイル名を書き換えて読み込んで**しまえば、アンシーリングして**その中身を取得できてしまう**

- この攻撃は、**シーリングデータのインデックス**（ファイル名等）による**参照**が**信頼不可能領域であるOS**で**行われる**事が**根本原因**である

シーリング再生攻撃実践

■シーリング再生攻撃実践課題
SGX-Vaultにおいて、**初期化処理後**に**何らかのパスワードを登録**し**シーリング**する（これを**ユーザAの秘密情報**とする）。
その**シーリングデータ**を**別の場所に退避させた後**、**ユーザBとして新たに初期化**する。

この時、**ユーザBのシーリングデータ**を、退避しておいた**ユーザAのシーリングデータ**で**置き換え**、**ユーザAが登録したパスワード**を**漏洩**させよ。

但し、簡単のためユーザA・Bのマスターパスワードはいずれも同一であるとする。

シーリング再生攻撃の対策

- LinuxのSGXSDKのv2.7くらいまでは、この攻撃を防ぐために**PSE**の提供する**モノトニックカウンタ**を使用できた
  - アクセスする度にカウントがインクリメントされる、信頼可能なカウンタ。このカウンタをシーリングデータに含める事で、古いカウントであるか否かを判定し、再生攻撃を防ぐ事が出来る

- しかし、v2.8以降**モノトニックカウンタ**（どころか**PSE**）が**Linux-SGXから廃止**された[17]ため、**この方法を用いる事は出来ない**
  - SGX Failのセクションでも述べた、Intelのずさんな無責任さが開発者の大幅な負担となる例

シーリング再生攻撃の対策実践（1/2）

■シーリング再生攻撃対策実践課題
SGX-Vaultについて、シーリング再生攻撃に対する防御策を導入せよ。

ただし、対策のためにリモートユーザ（SP）側で何らかの情報を
保持しても良いものとする。

シーリング再生攻撃の対策実践（2/2）

■シーリング再生攻撃対策実践課題のヒント・要件

- Enclave内で生成した乱数をシーリングデータに含め、SPにその乱数を暗号通信で送信し、SPはアンシーリング時にその乱数を送信する事でデータの鮮度を検証できる

- 本来はマスターパスワードがシーリング再生攻撃への対策にもなっているが、あくまで前述の通りマスターパスワードによる対策は行わないものとする

本セクションのまとめ

- SGXに対する様々な攻撃の一部を、その手法や性質に基づいて分類し、各攻撃についての簡単な解説を行った
- また、SGX-Bleedやシーリング再生攻撃について攻撃と防御の実践を行った
- その他の攻撃についての解説や実践は、後続のセクションで実施する

参考文献（1/3）

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

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

[3]”SGX Security – SGX 101”, 2023/7/8閲覧, https://sgx101.gitbook.io/sgx101/sgx-security

[4]”Hacking in Darkness: Return-oriented Programming against Secure Enclaves”, Jaehyuk Lee etal., https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-lee-jaehyuk.pdf

[5]”Teaclave SGX SDK”, GitHub, https://github.com/apache/incubator-teaclave-sgx-sdk

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

[7]”SGX-Bomb: Locking Down the Processor via Rowhammer Attack”, Yeongjin Jang et al., https://dl.acm.org/doi/10.1145/3152701.3152709

参考文献（2/3）

[8]”Plundervolt: Software-based Fault Injection Attacks against Intel SGX”, Kit Murdock et al., https://plundervolt.com/doc/plundervolt.pdf

[9]”PLATYPUS: Software-based Power Side-Channel Attacks on x86”, Moritz Lipp et al., https://platypusattack.com/platypus.pdf

[10]”Towards TEEs with Large Secure Memory and Integrity Protection Against HW Attacks”, Pierre-Louis Aublin et al., https://systex22.github.io/papers/systex22-final15.pdf

[11]”Scaling Intel SGX”, Wenhao Wang, https://heartever.github.io/files/scalable_sgx_public.pdf

[12]”Software Grand Exposure: SGX Cache Attacks Are Practical”, Ferdinand Brasser et al., https://www.usenix.org/system/files/conference/woot17/woot17-paper-brasser.pdf

[13]”Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing”, Sangho Lee et al., https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-lee-sangho.pdf

参考文献（3/3）

[14]”[SpectrePrime] [MeltdownPrime] とCPUのサイドチャネル攻撃 ～Evict-Time/Prime-Probe/Flush-Reload～”, 2023/7/13閲覧, https://milestone-of-se.nesuke.com/sv-advanced/sv-security/cache-side-channel/

[15]”SGXPECTRE: Stealing Intel Secrets from SGX Enclaves via Speculative Execution”, Guoxing Chen et al., https://yinqian.org/papers/eurosp19.pdf

[16]”Leaking Uninitialized Secure Enclave Memory via Structure Padding (Extended Abstract)”, Sangho Lee and Taesoo Kim, https://arxiv.org/pdf/1710.09061.pdf

[17]”Supports for Intel SGX SDK version 2.8”, GitHub, https://github.com/intel/sgx-ra-sample/issues/38

[18]”Training Solo: On the Limitations of Domain Isolation Against Spectre-v2 Attacks”, Sander Wiebing et al., https://download.vusec.net/papers/trainingsolo_sp25.pdf

[19]”Indirect Target Selection”, Intel, https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/indirect-target-selection.html