6. TDXのメモリ保護とその他追加機能

Ao Sakurai

2026年度セキュリティキャンプ全国大会
L1 – 暗号・CVMビルド&スクラップゼミ

本セクションの目標

- 本スライドでは、TDXにおけるメモリ保護機能についての詳細と、その他追加的なTDXの拡張機能について解説する

TDXにおけるアクセス制御の概要

TDXにおけるアクセス制御

- 以前説明の通り、TDXはIntel VTを拡張しながら実装された側面も含むため、TDXでは**VMXを活用**しながら**TDのメモリ分離を行う**

- よって、従来型のVMと同様、SMM、HV（ハイパーバイザ）、TDX Module、他のVM/TDといった、他のセキュリティドメインのメモリに**TDからアクセスできないのは従来と同様**である

- これは、TDで保護される形で潜むマルウェアが、そのマシンの他の要素を破壊する事が（TDX自体に脆弱性が無い限りは）できない事を意味し、**安全性の観点で重要な要素**である

TDXにおけるアクセス制御

- しかし、これまでも再三説明の通り、TEEであるTDXではHVを信頼する事ができないため、TDX ModuleがTDのプライベートメモリのアドレス変換を実施するという違いがある

- 同時に、TDXはホスト側の特権ソフトウェア、破損したデバイス、ホスト上の悪意のある管理者からメモリを保護する事が可能であり、これはTEEを名乗る上では必須の機能であると言える

- TDXでは、この保護を**アクセス制御**と**暗号学的分離**によって実現している

TDXにおけるアクセス制御

- アクセス制御の方は、同一マシン上の他のセキュリティドメインがTDにアクセスする事を防ぐものである
  - Arm TrustZoneのような古いTEEにも必ず搭載されている機能であり、TEEにおいては必須中の必須であると言える
- それに加え、TDXでは暗号学的分離、つまりはそのTD専用の暗号鍵（**TDエフェメラル秘密鍵**）で**TDのメモリを暗号化**する
  - これにより、悪意のあるDMAデバイスや、メインメモリに直接アクセスできる攻撃者が、TDのプライベートメモリを直接読み取り・破壊する事を防げる
- メモリ暗号化機能はIntel SGXやAMD SEV等でも搭載されており、TrustZoneには存在しないものの、現代のTEEにおいてはほぼ必須の機能であると言える

メモリのパーティショニング

メモリのパーティショニング

- アクセス制御と暗号学的分離の根幹となる仕組みとして、TDXではTDX有効化時、物理メモリ空間全体が**通常メモリ**と**セキュアメモリ**に分割される
  - 英語ではそれぞれNormal Memory, Secure Memory

- これらの概念は調べた限りではIntel公式の概念ではなく、Demystified論文[1]が独自に言及しているようであるが、概念理解の上で明快な分類であるため、本ゼミでもこれらを用いて説明する

メモリのパーティショニング

- セキュアメモリは、いわばTDXにおいて守られるべきコンポネントが配置される**保護されたメモリ**で、TDプライベートメモリ、vCPUステート、TDメタデータ等がこれに属する

- ただし、論文内での使い方を見ると、あくまでもTDX Moduleが管理するセキュアな領域、言い換えればTDエフェメラル秘密鍵で暗号化される対象を厳密には指しているようである
  - よって、TDXグローバル秘密鍵で暗号化されるTDR等や、専用のTME-MK鍵で暗号化されるSEAM Rangeは含まない可能性が高い

- とは言え論文内でもそこまで厳密な用語の使い方はしていないため、「**とりあえずTDXで保護されている領域**」くらいの感覚で良い

メモリのパーティショニング

- 通常メモリはTDXによって**保護されない領域**であり、これにはSEAMモードで動作しない全ての非TDコンポネントがまず含まれる
  - 例：従来型VM、ホストOS、ホストプロセス
  - 以前のスライドで説明した非階層型分離モデルに則り、これらのSEAMモード外コンポネントは、**その特権に関わらずセキュアメモリにアクセスする事はできない**

- 加えて、**共有メモリ**としてマークされたメモリも**通常メモリに含まれる**
  - 共有メモリは以前のスライドの通り、ネットワーク、I/O、DMA等、HVによる協力が必要な際に使用される、HVによってもアクセス可能なメモリ
  - 共有メモリはTDDエフェメラル秘密鍵では暗号化されないため、もし秘密情報を載せる場合には**TD内で暗号化してからその暗号文を載せる**必要がある

メモリのパーティショニング

- このようなメモリの分離を実現する上でのメモリアクセスのチェックは、プロセッサ内部のコンポネントである**メモリコントローラ**（MC）により実施される
  - MCはCPUのUncore上に存在するコンポネントである

- メモリ暗号化を司るMK-TMEエンジンもUncore上に存在するが、MCの外側（よりCPUコアに近い側）に位置しているため[2-4]、あくまでもそれぞれ別の役割を持つと言える

メモリのパーティショニング

- TD構築の際には、信頼不可能なHVは、セキュアメモリの一部となるメモリページを通常メモリから選択する
  - そして、TDX Moduleがこれらのページを逐次セキュアメモリに移動させる
- 同時に、TDX ModuleはPAMTを保持する事で、セキュアメモリの構築の整合性チェックを行う（真に特定のTDにのみ占有的に使用される事を保証する）のは前回記事で説明した通り
- セキュアメモリページの追加周りの説明はAttestationのスライドにて詳述する

メモリの機密性とHKID

TME-MKによる保護

- TDのプライベートメモリやメタデータ、そしてSEAM Rangeといった要素の暗号化には、**TME-MK**というHWベースの暗号機能が使用されるのは以前のスライドでも説明した通り
  - TME-MKの暗号処理は前述の**MK-TMEエンジン**で実際に行われ、このエンジンがMCを通過するデータを自動的に暗号化・復号する事で、透過的にTDXにおける保護対象が暗号化・復号される
- TME-MKでは、メモリ暗号化方式として**128bit AES-XTS**と**256bit AES-XTS**を提供している
- ただし、TDXにおいては明文化されていない未知の仕様が無い限り、**前者の128bit AES-XTSにのみ対応**している[5]

TME-MKによる保護

- MK（Multi-Key）の文字の通り、**複数の暗号鍵を生成・割当**する事が可能である

- TDXにおいては各TD、TDXグローバル秘密鍵、SEAM Range暗号化鍵それぞれに別々の鍵を割り当てて暗号学的に隔離する、という使い方がされている

TME-MKによる保護

- TME-MKによる暗号処理を含める**メモリへの書き込み**自体は**キャッシュラインサイズ**（64B）**単位**で行われる一方、**TME-MK鍵の割り当て**は**メモリページ**（4kB）**単位**で行われる

- キャッシュライン粒度での書き込みの際、TME-MKは物理アドレスに埋め込まれた**HKID**を抽出・参照する事で、対応する鍵による暗号化を実施する
  - HKID：Host Key Identifier

- この**TME-MK鍵**は**MK-TMEエンジンの内部メモリで管理**され、**外部に鍵が露出する事はなく**、かつこの鍵はこのHKIDによってのみ参照する事ができる

HKIDの概要

- 新しいTDを構築、つまりは新しいHKIDを使用する際には、まず**HVが未使用のプライベートHKIDを選択**する

- そして、HVが選択したこのHKIDに関連する新しい**TME-MK鍵を生成**するよう、**TDX Module**が依頼を投げる

HKIDの概要

- これにより、MK-TMEエンジン内でHKIDとTME-MK鍵のタプルが生成され、これはエンジン内の**KET**内で保持される
  - KET：Key Encryption Table

- そして、TDX ModuleはそのTDに対してこのHKIDを紐づける（**物理アドレスに埋め込む**）事により、間接的に**HKIDとTME-MK鍵のタプルをTDに紐づける**事ができる

HKIDの概要

- まとめると、TD作成時にはまずHVがHKIDを選択し、その後TDX Moduleが受け取ったHKIDと未使用のTME-MK鍵を紐づけるようにMK-TMEエンジンに要請しバインドする
  - そして、そのTDページの物理アドレスには、そのHKIDを埋め込む
- その後は、埋め込んだHKIDを参照する事で、適切なTME-MK鍵が選択されページ暗号化や復号に使用される
- この暗号化で存在する事で、コールドブート攻撃のような機密性に対する**物理攻撃により直接平文を観測される事を防げる**
  - コールドブート攻撃：マシン終了後にDRAMを取り出して冷却し、電荷揮発を遅らせつつ解析器で内容を盗聴する攻撃

HKIDの詳細

- HKID空間（後に詳述する、物理アドレス上位ビットで表現可能な取り得るHKIDの範囲）は、ブート処理中に一度、**プライベートHKID**と**共有HKID**の2つの範囲に**パーティショニング**される

- プライベートHKIDはSEAMモードのソフトウェア、つまりはTD、TDX Module、P-SEAM Loaderといったもののみが、プライベートHKIDに紐づけられた鍵で内容が暗号化されたメモリを読み書きできる

HKIDの詳細

- 一方で、共有HKIDに関連付けられた鍵は、レガシーVMのメモリやホストカーネル等、SEAMモード以外のメモリを暗号化するために使用できる

- また、共有メモリの暗号化にもこの共有HKIDに紐づいた鍵が使用される
  - Intel公式文書[6]に以下のような記述がある事から特定可能：「TD Shared memory is designed to hold content accessible to the TD and the host software (and/or other TDs). TD shared memory may be encrypted using MKTME keys managed by the VMM.」
  - VMM（HV）が管理できるのは共有HKIDのみである事を併せて考えると、そのように結論づける事ができる

HKIDの詳細

- プライベートHKIDを使うにはまず、HVがTDの構築を要求する際に、HVによりそのTD用のプライベートHKIDを、未使用のプライベートHKIDから選択する

- それを受けて、TDX Moduleは**PCONFIG**命令を使用する事で、TME-MKに対しHKID用の一意なランダム鍵を生成するように要求する

- この鍵こそがこれまでにも何度も登場している**TDエフェメラル秘密鍵**である
  - この呼び方はDemystified論文固有のものではなく、実際にIntelの公式文書[5]においてもそのような呼び方がされている

HKIDの詳細

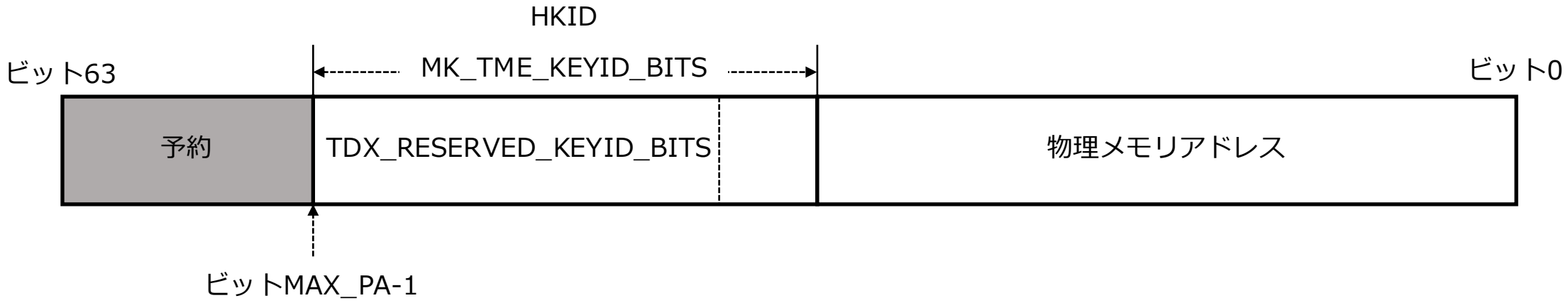
- この鍵はTDの全てのプライベートメモリとメタデータの暗号化に使用され、TME-MKの外部に公開される事はない

- また、この鍵はTDX Module起動前に割り当てが行われるTDXグローバル秘密鍵や、TDX Module自体が載るSEAM Range用暗号化鍵とは**別物**であると言える
  - TDエフェメラル秘密鍵はあくまでもTDX Moduleにより割り当て・管理されるものであるため

- このPCONFIG命令により生成された<HKID, key>のバインディングはTDが有効である限り有効である

物理アドレスにおけるHKIDのレイアウト

- 以下に、このHKIDに関連付けられた物理メモリアドレスのレイアウトの図を示す（図は[1]内の図を参考に作成）：



物理アドレスにおけるHKIDのレイアウト

- 前図を見ると分かる通り、文字通り**物理アドレスの特定のビット範囲**を**そのままHKIDの表現に用いている**事が分かる

- この物理アドレス上におけるHKIDの設定に関して、ブート時にMSR（モデル固有レジスタ）である**IA32_TME_ACTIVATE**の以下のビットがそれぞれ指定される：
  - **MK_TME_KEYID_BITS**：HKIDに使用するビット数
  - **TDX_RESERVED_KEYID_BITS**：プライベートHKIDに使用するビット数

- ちなみに、TDXマシン構築時にBIOS設定において「TME-MT/TDX key split」のような項目があるが、この項目はこのMSRの設定に絡んでいるものであると思われる
  - TME-MT：TME Multi-Tenantの意。TME-MKと同義

物理アドレスにおけるHKIDのレイアウト

- アクティベートされた（使用中の）プライベートHKID及び共有HKIDの数の取得は、IA32_MKTME_KEYID_PARTITIONING MSRの読み取りにより行える

- 前述の**TDX_RESERVED_KEYID_BITS**は、前図からも分かる通り**MK_TME_KEYID_BITSの一部**である
  - よって、前者は後者の部分集合であると言える

- Intelの公式文書[8]ではTDX_RESERVED_KEYID_BITSをMK_TME_KEYID_BITSが超えてはならないとしており、これを見ると同値にする（全てプライベートHKIDにする）事は可能らしい？
  - 共有メモリ含むホスト側は一切暗号化しないという割り切った判断をするのであればもしかするとそのような設定も可能かも知れないが、あくまで実機確認していないので推測に過ぎない点には注意

HKID数の算出方法

- 具体例としてMK_TME_KEYID_BITSが6、TDX_RESERVED_KEYID_BITSが4の場合を例に、共有HKIDとプライベートHKIDの数の算出方法を説明する

- 1つややこしい点として、**プライベートHKID側**は**上位ビットから占有していく**という仕様になっている
  - よって、単純にTDX_RESERVED_KEYID_BITSが4bitだから16個、共有HKIDはHKID総数の6bit分からそれを差し引いて48個、といったような計算はできない

HKID数の算出方法

- 実際には、プライベートHKIDがまずTDX_RESERVED_KEYID_BITSが示す上位4bit分となる

- そして、共有HKIDはMK_TME_KEYID_BITSからTDX_RESERVED_KEYID_BITSを引いた、つまりプライベートHKIDが予約した余りの6 - 4=下位2bitとなる

HKID数の算出方法

- 結果、共有HKIDは単純に下位2bit分を占有する事になるため、
  そのまま $2^2 = 4$ が個数となる

- 一方、プライベートHKIDは上位ビットであるため、HKID全体個数
  から共有HKIDに使われている個数を引き算してやる事で
  個数を算出できる

- この例の場合は$2^6 - 2^2 = 60$となるため、プライベートHKIDは
  60個となる

HKIDによるアクセス分離のまとめ

- このようにして決定される割り当て数をそれぞれ上限とし、HVやTDX ModuleはメモリページのPA上位ビットにHKIDをセットする事で、それぞれのTME-MK鍵によりメモリ暗号化できる
  - この時HVは共有HKIDのみを使用できる一方で、TDX Moduleは共有HKIDとプライベートHKIDの双方を使用する事ができる

- なお、SEAMモード以外で実行されるSWがプライベートHKIDを持つ物理アドレスを通してメモリアクセスを試みると**例外が発生**する

メモリの完全性

TD Owner Bit

- TDXは、ソフトウェアレベルの攻撃（及び、一部のHWレベルまたは物理的攻撃）に対する、TDプライベートメモリのメモリ完全性の保護も提供する

- これは、悪意のあるHVやDMAデバイス等の、SEAMモード外の主体がTDのプライベートメモリに書き込む（暗号化されているメモリに無理矢理上書きするイメージ）可能性を排除できないためである

- このTDXにおける完全性保護及びメモリの隔離保護の上で欠かせないのが**TD Owner Bit**という存在である

TD Owner Bit

- TD Owner Bitは、キャッシュラインサイズ（64B）の「セグメント」に関連付けられたECCメモリ上（具体的にはその中の1bit [9]）に格納される
  - **ECCメモリ**：誤り検出訂正のために、通常はデータメモリ64bitごとに8bitが割り当てられる追加のメモリ

- 違和感があるかも知れないが、ECCビットは最高で64**bit**単位の細粒度で管理できるのに対し、TD Owner Bitは64**B**単位と、それよりも大分粗い点に注意
  - これは、前述の通りTME-MKの暗号化等の処理単位が64Bであるためそちらに合わせているためであると考えられる
  - つまり、**ECCビット自体は64bitデータ幅ごとに割り振られている**が、あくまでも**TD Owner Bitは64Bごとにしか使っていない**という事である

TD Owner Bit

- 物理ページをセキュアメモリの一部にするには、まずそのページに対応する全てのTD Owner Bitを有効、つまりは1にする

- そして、**SEAMモードの主体**（TDやTDX Module等）が**セキュアメモリ**のメモリセグメントに**書き込む**と、MCは**対応するTD Owner Bitを1にする**

- 反対に、**非SEAMモードの主体**（HV等）がセキュアメモリに書き込むと、**対応するTD Owner Bitは0にクリア**される

TD Owner Bit

- 読み取りに関しては、SEAMモードで実行中のプロセスのみが、TD Owner Bitが1にセットされたキャッシュラインを読み取る事ができる
  - **非SEAM**による読み取り要求がこのようなキャッシュラインを読み取ろうとすると、**オール0列を受け取る事**になる

- この仕組みは、ソフトウェアがメモリの暗号文を読み取ってその変化等を観察する事で秘密情報を推測しようとする、**SWレベルの暗号文サイドチャネル攻撃の対策として重要**である

TD Owner Bit

- 実際、AMD SEV-SNPでは元々この暗号文隠蔽機能が無かったため、CipherLeaks[11]という攻撃が実証されてしまっている
  - 今ではSEV-SNPも**Ciphertext Hiding**という機能により外部からの暗号文の読み取りを阻止できるようにしている

- また、SGXでは**Abort Page Semantics**という仕組みにより、権限外読み取りには0xFF列を返すようになっており、こちらは元々暗号文サイドチャネルに対して耐性を有している

TDXにおけるメモリ完全性保護機能

- 完全性保護の要の1つであるTD Owner Bitについて説明したので、次はTDXにおけるメモリ完全性保護の詳細について説明する

- TDXにおいては、メモリ完全性の保護のために以下の2つのメカニズムを提供するには以前のスライドで述べた通り：
  - **Logical Integrity**（論理的完全性；**Li**）
  - **Cryptographic Integrity**（暗号学的完全性；**Ci**）

Logical Integrity

- Liは、先程説明した**TD Owner Bitを利用し**て、SWレベルでの不正な書き込みに対する完全性保護を提供する

- 前述の通り、非SEAMモードの主体からセキュアメモリに書き込みが発生すると、対応するTD Owner Bitは0になる

- このTD Owner Bitが0になっているセグメントをTDやTDX Moduleといったモード主体が読み出すと、そのセグメントは**汚染（Poisoned）**されているとしてマークされる

Logical Integrity

- 読み出し側がTDである場合、この汚染マークにより直ちにTD Exitすると共に、**MCE**または**MSMI**という特殊な例外や割り込みが発出される[6]
  - MCE : Machine Check Exception
  - MSMI : Machine-check System Management Interrupt

- この時、TDX ModuleはこのTD Exitをキャプチャし、TDを**致命的（Fatal）ステート**とする事ができる

Logical Integrity

- 一方、読み出し側がTDX Moduleである場合、TDX Moduleとプロセッサ内のTDXハードウェアが無効であるとマークされる

- その結果、TDX Moduleは中断不能な強制終了（Unbreakable shutdown）が為され、これは同時に**SEAMシャットダウン**を誘発する[6]

- SEAMシャットダウン以降、**全てのSEAMCALL**はVMFailInvalidエラーを引き起こし**異常終了**する事になる

Logical Integrity

- ただし、このLiという機能単体では、RowHammer攻撃のような**ビット反転攻撃**を**防ぐ事はできない**
  - RowHammerでビット反転を試みた場合、書き込みはしていない（過剰電荷で電磁気学的に反転させている）ので、TD Owner Bitは1のままとなる

- そもそもECCビットがあるRDIMMの時点でRowHammer攻撃は通用しないのではという話もあるが、ECCを迂回するようなタイプのRowHammer攻撃もいたちごっこ的に登場している点には注意

- その意味でもそのような攻撃をより強固に対策するには、次に説明する**Ciが有効**である

Cryptographic Integrity

- CiはLiよりも高度なメカニズムであり、TD Owner Bitでの管理に加え、まず書き込み時に**キャッシュラインごと**に**MAC**（メッセージ認証符号）を**計算**する
  - このMACは、システム初期化時にプロセッサにより生成される128bitのMACキーを使用して計算される
  - このMACにはSHA-3-256（KECCAK[512]）が使用される

余談だが、KECCAKはアメリカのNISTのコンペティションに応募された暗号学的ハッシュ関数で、これが後にSHA-3と呼ばれるようになった。SHA-3-256の256は出力ハッシュ長、KECCAK[512]の512は、その長さの2倍として設定されるキャパシティというもののサイズを表しているようである。
詳細については、SHA-3のWikipedia記事を参照の事。

Cryptographic Integrity

- MAC長自体は256bitであるにも関わらず、鍵長が128bitというのは直感的には違和感があるかも知れないが、これ自体は割と普通によく行われる事のようである

- NISTの文章[12]でも、以下のように鍵長とMAC長が一致するとは限らない前提である記述がされている：
  - In Steps 7 and 8, the final CBC output block is truncated according to the MAC length parameter that is associated with the key, and the result is returned as the MAC.

Cryptographic Integrity

- このMACは以下の要素に対して計算される：
  - 暗号文（キャッシュラインの暗号化されたコンテンツ）
  - AES-XTS暗号化に使用されるTweak値（外部パラメータ）。AES-XTSの場合は物理アドレスがTweak値となる
  - TD Owner Bit
  - 128bitのMAC鍵（鍵なのでこれだけはメッセージではないはず。ただ、[6]の書き方的にも、対象にMAC鍵自身が入るのか若干不明瞭）

- 書き込み時、この256bitのMACは**28bitに切り詰め**られてから、対応する**ECCメモリにこれを格納**する
  - ECCメモリは、1セグメント（データ64bit幅に対応）辺り通常8bitであるため、セグメントを跨ぐ形で配置されるのであろうと推測される

Cryptographic Integrity

- Ciの処理単位もTD Owner BitやTME-MKの単位と同じく64Bであるため、28bitもあれば互いに干渉を起こす事は無いと考えられる
  - これについての詳細な議論は次ページで実施
- 読み出し時にも同じくMACを計算し、もし**不一致があった場合**には**改竄が発生したと見なす**
- その際にはLiの時と同様、MCEやMSMI、TDX ModuleのUnbreakable shutdown等を発生させる
  - 参考文献[6]のTable 16.2を参照

Cryptographic Integrity

- この256bitのMACを28bitに切り詰めるという仕組みは、一見する安全性を損なうようにも見える

- しかし、以下の理由から**狙って衝突を引き起こして完全性チェックを迂回**する事は**神でもない限りは不可能**と言って良い：
  - **MAC鍵はプロセッサを出ず**、衝突を狙うなら**総当たりさせるしかない**
  - 28bitは2^{28}、つまり**約2億6800万通りを表現**できる
  - 一回でも間違えれば**即座にMCEやSEAMシャットダウンが発生**する
  - そもそもECCビットを読み取る事はTDXの脅威モデル外である**物理的攻撃を行わない限りは不可能**

Cryptographic Integrity

- 反面、メモリ上のデータ（正しい暗号文）とECCビットの双方でロールバックするような強力な攻撃者に対しては無力である
  - しかし、2026年5月時点では少なくともそのような攻撃は研究レベルでは実現されていない

- Battering RAM[9]というインターポーザベースの物理攻撃の論文では、論文で行っているデータの再生に加えてECCビットまで再生するのはかなり高度なインターポーザを必要としていると述べている
  - この事からもその難易度の高さが窺える

TDXにおける追加機能

TDXにおける追加機能

- TDXは、最初にTDX 1.0としてリリースされた後、TDX 1.5やTDX 2.0としてバージョンアップしている
  - Demystified論文[1]はTDX 1.0の時代に書かれたものだが、バージョンアップに伴い様々な機能が追加で搭載されている

- そのような機能として、主に以下のような機能が存在する：
  - **ライブマイグレーション**
  - **TDX Connect**
  - **TD Partitioning**

ライブマイグレーション

ライブマイグレーション

- **ライブマイグレーション**：VMを停止させる事なくマイグレーション（ベアメタルを移動等）する事
  - TDXに限らず仮想化において一般的に行われる手続きである
- これにより、既にVMが稼働しているベアメタルマシンを停止する必要がある時にも、VMを中断する事なく他のベアメタルマシンに移してからメンテナンス処理を行える
  - HWアップグレード、SWパッチ、ロードバランシングの際等
- これは特に大量のマシンとVMを管理するクラウドサービスプロバイダ（CSP）にとっては非常に重要な機能である

ライブマイグレーション

- ただし、TDXを含むCVMではHVを信頼不可能と見なすため、通常のVMのライブマイグレーションのようにはいかない

- また、テナントはマイグレーションポリシを定義しそれを強制する事で、意図しないマイグレーションを阻止できる必要もある
  - テナント：TD所有者、つまりはリモートからTDを利用するユーザ

- 基本的に、CVMのライブマイグレーションの実現のための追加機能は各TEEベンダが追加で実装し提供する必要がある
  - TDXにおいてはTDX1.5にてこのライブマイグレーション機能が提供開始されている[13]

ライブマイグレーション

- このような課題を解決するために、TDXでは**サービスTD**という特別なTDを導入している
  - このサービスTDは信頼する前提であるため、**狭義のTCBの一部**である
- TDX Moduleを拡張してライブマイグレーションを担当する事も可能ではありますが、それではSEAM VMX rootである特権TCBがさらに肥大化する事になる
  - この肥大化は脆弱性の混入リスクも高め、そしていざ混入してしまった場合には、特権であるが故により甚大な被害をもたらす可能性が高い
- 代わりにこのサービスTDというSEAM VMX non-rootで動作するユーザレベルTCBに任せる事で、より特権的なTDX Moduleの脆弱性のリスクを下げ、同時に柔軟性も上げる事をIntelは選択した

ライブマイグレーション

- これと似たような措置は先行するIntelのTEE実装であるIntel SGXにおいても存在する

- SGXでは、例えばAttestation関係のような、マイクロコードで実装するには複雑すぎる処理を、**Architectural Enclave（AE）**というIntel署名付きの特別なEnclaveに任せている

- マイクロコードとTDX Moduleという違いはあれど、より低い特権の主体にオフロードするという意味では、似た考え方であるとも言える

MigTD

- その中で、**マイグレーションTD（MigTD）**というものが
  ライブマイグレーション専用に設計されたサービスTDとして存在する
  - これはDemystified論文だけでなく現行のIntel公式仕様書[6][14]にも
    しっかりと登場している

- MigTDは完全にTDX Moduleベースで管理されるものであり、
  Intel公式仕様書によっても（狭義の）**TCBの一部**であると
  明記されている[15]

MigTD

- MigTDの動作が正しいかの保証は、ソースと宛先のMigTDの間で相互Remote Attestation（RA）を実施する事で担保する[11]
- 具体的には、実際にQuoteを相互に生成し互いに検証している[11]
- RAについての詳細は後のスライドで解説する

MigTD

- その上で、細かい説明とはなるが、MRTD（そのTDの初期メモリ配置に対するハッシュ値）等をどのように検証するかは、マイグレーションポリシで指定できる[11]
  - 例えば、宛先MigTDのMRTDがソースと同じ場合にのみOKとする、あるいは特定の測定値のみ許可する、といった指定が可能[11]

- MigTDのソースコードは、こちらのIntel公式リポジトリから確認する事ができる

TDライブマイグレーションの流れ

- ライブマイグレーションセッションは、TDX ModuleとMigTDの制御下で進行する

- CSPが制御する信頼不可能なHVは、暗号化されたTDのアセット（TDメタデータ、CPUステート、プライベートメモリ）をネットワーク経由で転送する事のみを担当する

- これらのアセットは、MigTDとTDX Moduleのみがアクセス可能であるマイグレーションセッション鍵（**MSK**）によって保護される

TDライブマイグレーションの流れ

- MigTDは移行元（ソース）と移行先（宛先）の双方に存在し、それぞれソースTDと宛先TDにバインドされる
  - その上で両MigTDは前述の通り相互RAを実施する

- このRAによりプラットフォームがマイグレーションに適していると判断できたら、MigTD間でTLSセキュアチャネルが確立される

- このRAからセキュアチャネル確立までの一連の手続きは、**RA-TLS**という手法により行われる
  - **RA-TLS**：RAをTLSにバインドし、真に通信相手がRA済みである事を確信できる状態にする「Attested-TLS」と呼ばれる手法のIntelによる一実装

TDライブマイグレーションの流れ

- セッション確立後、ソースMigTDはMSKを生成し、このセキュアチャネルを通して宛先MigTDにMSKを送信する

- その後、ソースTDX ModuleはTDのアセットをエクスポートしてMSKでそれを暗号化し、宛先TDX Moduleは同じくMSKで復号し宛先TDにインポートする

TDライブマイグレーションの流れ

- ソースTDと宛先TDはHKIDが独立して割り当てられているため、両TDそれぞれ異なる暗号鍵で保護される事が保証される
  - 違うマシンにマイグレーションする場合にはそもそもマシンが異なるため、当然に鍵はこの場合も別々となる

- ちなみに、AMD SEV-SNPではマイグレーションエージェントというものがMigTDに似たコンポネントとして存在する

TDX Connect

TEEにおける外部コンポネント

- 言うまでもなく、コンピュータは様々なコンポネントから構成されている

- しかし、Confidential Computingの文脈においては、**異なるベンダにより製造**された**各コンポネント**は**互いに信頼する事はできない**

- この件でよく槍玉に挙げられていたのが「**TEEからはGPUを使用する事ができない**」というものである

TEEにおける外部コンポネント

- SGXやTDXのようなCPUのTEEは、それ単体ではあくまでもCPU及びそれと接続されているDRAM上の特定の区画のみを保護するものである

- 逆に言えば、NVIDIAやAMDが製造したGPUコアやVRAMをCPUのTEEにより保護する機能は当然に存在しない

GPU TEE

- そのような中、NVIDIAはHopperアーキテクチャGPU（H100、H200）からGPU側のTEE機能である**NCC**（NVIDIA Confidential Computing）を実装し提供開始している

- これは、CPU TEE側とはAttestationの上でセキュアチャネルを張り、秘密情報や指令の伝送を行う事で、CPUのTEEの範囲をGPUまで拡張する事に成功している

GPU TEE

- しかしこのHopperアーキテクチャのNCCは、CVMがGPUとデータをやり取りする際、一度データをCVMの共有メモリ上の**バウンスバッファ**にコピーしなければならない、という問題点がある

- 何故ならば、GPUはCVM側のプライベートメモリにはアクセスできないため、TDXで言う所の共有メモリ上のバッファを経由して相互にやり取りしなければならないためである
  - 少なくともHopper時点では、GPUはTDX等CVMの信頼境界内に組み入れてよいというセキュリティモデルになっていない

- これは、**効率的なI/O**を実現する上での**深刻な支障**となる

TDX Connect

- そこで、TDXはTDX 2.0において[13]**TDX Connect**を実装し、信頼可能と見なしたデバイスまで信頼を拡大する機能を搭載する

- これは、デバイスとTDXプラットフォーム双方に互換性のある PCI-SIGが定めた**TDISP**に、TDX側が対応するための TDX側拡張機能である
  - TDISP： TEE Device Interface Security Protocol

- これにより、デバイスとTDとで安全な通信路を張り、そして**Device Attestation（DA）**で**デバイスの信頼性を検証**した上で、デバイスに**直接TDプライベートメモリにアクセス**させ**効率的なI/Oを実現**できる

その他TDISP周りの状況

- ちなみに、NVIDIA側は**Blackwellアーキテクチャ GPU**（B200等）以降で**このTDISPに対応**している[16]

- また、AMD SEV-SNPでは**SEV-TIO**、Arm CCAでは**RME-DA**という機能がTDX Connect相当の機能として実装されている

- ただし、OS修正含め全てがアップストリームかというと2026/5時点では怪しく、完全に使えるようになるにはもう少し時間が必要であると予想される

TD Partitioning

CVMゲスト内での特権分離

- AMD SEV-SNPでは、**VMPL**（仮想マシン特権レベル）という概念が存在し、1ゲストCVMの中でさらにVMPL0～3の**4つの特権レベル**に**ゲストコンポネントを分割する事が可能**である

- また、Arm CCAではPlaneという機能が存在し、より特権の大きいPlane 0（P0）と、ハードウェアリソースが許す限り何個も生成可能な複数個のPlane 1以降（Pn）を生成する事ができる[17]

- これらの典型的な用途として、VMPL0やPlane 0のような**特権的なレベル**で**vTPM（仮想TPM）を動かす**というものがある
  - この特権的に隔離されたゲスト内vTPMにより、動的ファームウェアやゲストOS、ユーザワークロードといった**ランタイムワークロードのAttestationをゲストに侵害される事なく実現**できる

CVMゲスト内での特権分離

- 一方、TDXにおいてはこのような**ゲスト内隔離機能は存在しない**

- Attestation編のスライドで詳述するが、TDXにおいては**RTMRという測定レジスタが存在**し、最も典型的なユースケースであるvTPMの配置が必須ではない

- これにより、わざわざそのようなゲスト内特権分離を実装しなくて良いとIntelが判断したのではないかと推測できる

CVMゲスト内での特権分離

- しかし、特にTEEクラウドの提供に力を入れている事で有名なMicrosoft Azureは、OpenHCL等の「**パラバイザー**」と呼ばれる独自のゲスト内基盤（ファームウェア）を使いたがる傾向にある[18]
  - これは主にvTPMを多分に活用するものであるため、ゲスト内隔離機能が無ければ安全にvTPMを隔離できず、安全性に問題が生じる

- 他にも、ゲスト内でさらにTEE機能による強力な隔離を行いたいというニーズは、特に**マルチテナント環境**においてはリソース最適化の観点でも安全性の観点でも、自然に発生するものであると言える

TD Partitioning

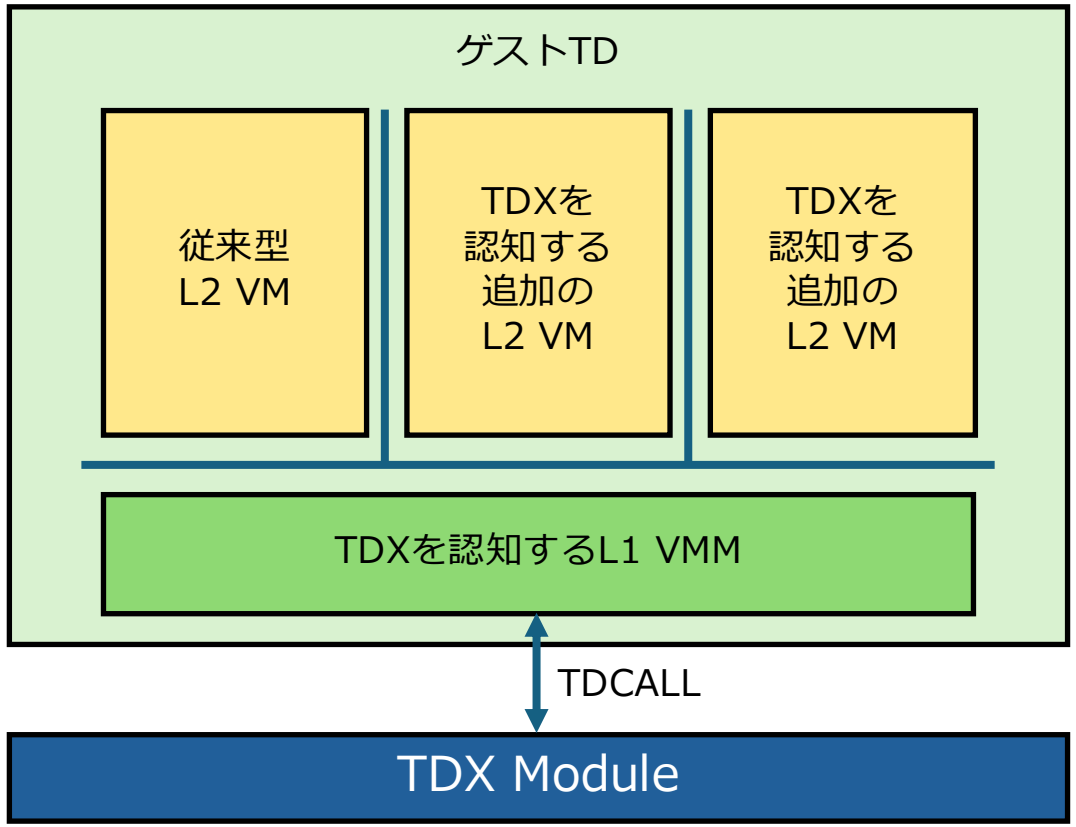
- そこで、TDXはTDの中でさらにVMを生成する、ネストされた仮想化機能を搭載する事を選択した

- この機能は**TD Partitioning**と呼ばれ、TDX 1.5にて実装された[15]

- TD Partitioningでは、ゲストTDの中でさらに以下のものを動かす事ができる：
  - TD内のネストされたHVとして機能するL1 VMM（として振る舞うL1 VM）
  - 1つの従来型L2 VM（TDX向けの改造不要）
  - オプションとして2つのTDXを認知可能な追加のL2 VM（TDX-enlightened L2 VM）

TD Partitioning

- TD Partitioningの概要図を以下に示す（図は参考文献[19]内の図を参考に作成）：



TD Partitioning

- Microsoft側参考文献[18]やIntel側参考文献[19]を見るに、AzureのTDXインスタンスでは、図2のL1 VMM部分のvTPM等を置き、実際にユーザに提供するゲストVMはL2 VMとして配置する構成になっている
  - これは至極理にかなった設計であると言える

- Microsoftはこの機能をIntelと共同開発したと言及しており[18]、これはAzure以外の、例えばベアメタル上でTDXを動かす一般ユーザが手を出せる代物であるとは考えにくい

- それでも、VMPLやPlaneのような機能をTDXで実現できるという意味で、画期的な機能の1つであると言える

まとめ

- TDXにおけるメモリ保護の詳細について解説した

- その他、新しいバージョンのTDXで追加搭載された各種追加機能について解説した

参考文献

[1] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel TDX Demystified: A Top-Down Approach. ACM Comput. Surv. 56, 9, Article 238 (September 2024), 33 pages. https://doi.org/10.1145/3652597

[2] “Integrity protected access control mechanisms”, https://patents.google.com/patent/US20220209933A1/en

[3] “Intel® Architecture Memory Encryption Technologies", Intel, https://cdrdv2-public.intel.com/679154/multi-key-total-memory-encryption-spec-1.4.pdf

[4] “EC-CFI: Control-Flow Integrity via Code Encryption Counteracting Fault Attacks", Pascal Nasahl et al., https://ieeexplore.ieee.org/document/10132915

[5] “Intel(R) Trust Domain Extensions White Paper”, Intel, https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf

[6] “Intel® Trust Domain Extensions (Intel® TDX) Module Base Architecture Specification”, Intel, https://cdrdv2.intel.com/v1/dl/getContent/733575

[7] “Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 4: Model-Specific Registers”, Intel, https://cdrdv2-public.intel.com/835765/335592-sdm-vol-4.pdf

参考文献

[8] “Intel® Trust Domain CPU Architectural Extensions”, Intel, https://cdrdv2-public.intel.com/733582/intel-tdx-cpu-architectural-specification.pdf

[9] “Battering RAM: Low-Cost Interposer Attacks on Confidential Computing via Dynamic Memory Aliasing”, Jesse De Meulemeester et al., https://batteringram.eu/batteringram.pdf

[10] “Error Correction Code (ECC) in DDR Memories”, 2026/3/11閲覧, https://www.synopsys.com/articles/ecc-memory-error-correction.html

[11] “CipherLeaks: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel”, Mengyuan Li et al., https://www.usenix.org/system/files/sec21-li-mengyuan.pdf

[12] “Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication”, Morris Dworkin (NIST), https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38b.pdf

[13] “Releases - confidential-computing.tdx.tdx-module”, GitHub, https://github.com/intel/confidential-computing.tdx.tdx-module/releases

参考文献

[14] “Intel® TDX Module Architecture Specification: TD Migration”, Intel, https://cdrdv2.intel.com/v1/dl/getContent/733578

[15] “Trust Domain Extension (TDX) Migration TD Design Guide”, Intel, https://www.intel.co.jp/content/www/jp/ja/content-details/733580/intel-tdx-migration-td-design-guide.html

[16] “CC on Blackwell - TDIS support #70”, GitHub, https://github.com/NVIDIA/nvtrust/issues/70

[17] “Arm CCA Extensions”, Arm, https://developer.arm.com/documentation/den0125/400/Arm-CCA-Extensions

[18] “OpenHCL: the new, open source paravisor”, Microsoft, https://techcommunity.microsoft.com/blog/windowsosplatform/openhcl-the-new-open-source-paravisor/4273172

[19] “Intel® Trust Domain Extensions (Intel® TDX) Module TD Partitioning Architecture Specification”, Intel, https://cdrdv2.intel.com/v1/dl/getContent/773039