4. TDXにおける仮想化

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

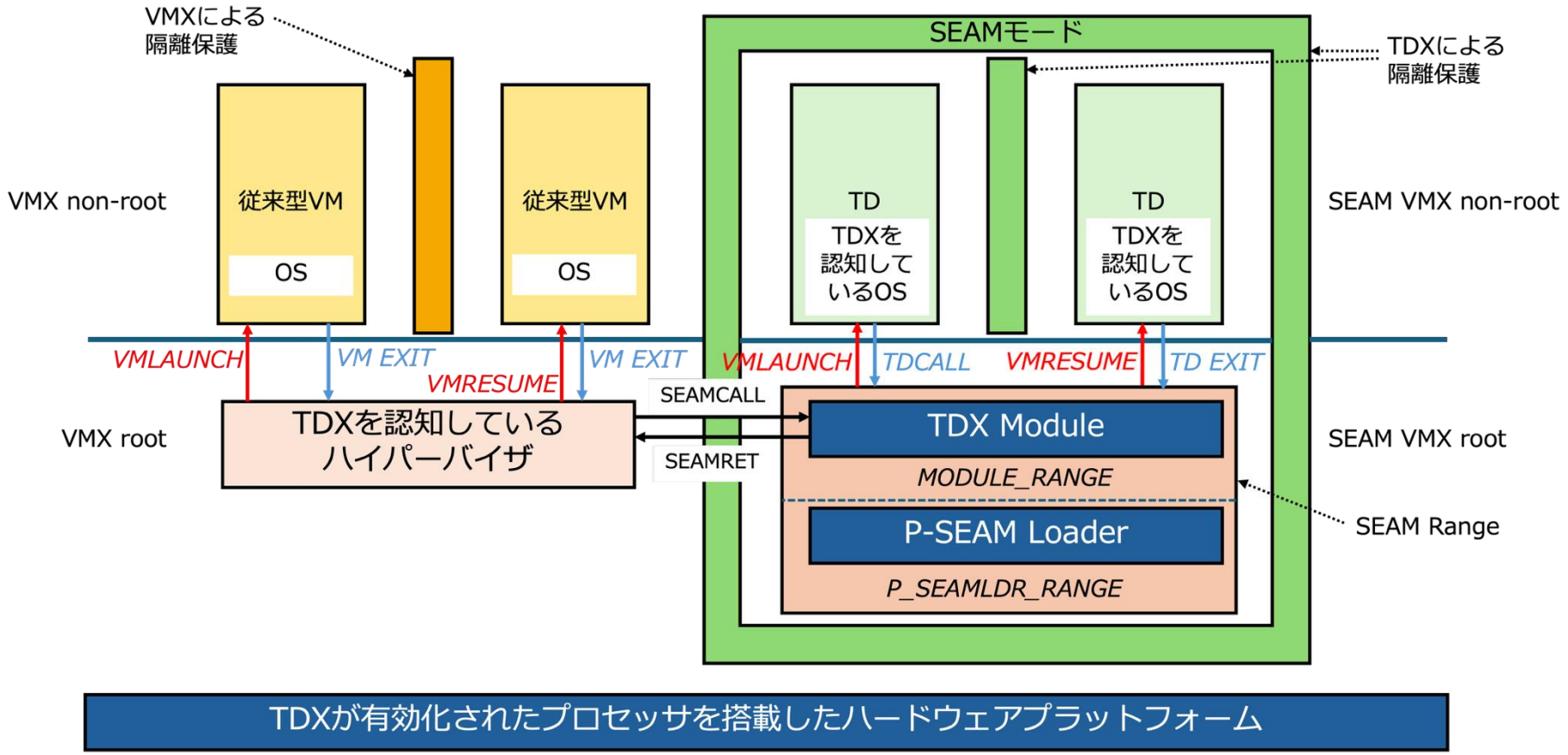
- 本スライドからは、TDXを構成する各要素についてより詳細に説明していく

- 本スライドでは、手始めにTDXにおける仮想化について詳細に説明する

TDXにおける基盤

TDXにおける基盤

- まず、仮想化周り観点でのTDXの全体像を以下の図に示す（図はDemystified論文[1]を参考に作成）：



TDXにおける基盤

- TDXを利用する上で必須なものと言えば、まず言うまでもなく**TDX対応プロセッサ**である
  - HW支援型仮想化、メモリ暗号化及び完全性保護、Attestationといった TDXの基礎となる各種アーキテクチャ機能を提供する

- 2つ目に、**TDX Module**も必須であり、これは以前説明の通り Intel TXTという技術が提供するIntel ACMを用いたロードプロセスにより、真にIntelが用意したものが使われる事が保証される
  - 若干専門的に言い換えれば、その真正性は**HW-RoT**（Hardware Root-of-Trust）で保証されているという事である

TDXにおける基盤

- TDX Moduleは、TDX対応プロセッサを活用しながら、TDXのセキュリティ保証を担保しつつTDの構築、実行、終了等を容易にする役割を持つ

- 例えばSGXでは、SGX専用の処理は何をするにも直接CPU命令を叩く必要があった
  - 例：Report構造体が欲しいのであれば、（SGXSDKによるラッパーがあるとは言え）ENCLU[EREPORT]命令を発行しなければならない

TDXにおける基盤

- 一方、TDXではゲストやハイパーバイザ（HV）がこのような
  コアのCPU命令を直接叩く必要はない

- あくまでもTDX Moduleが提供するインタフェースを呼び出すだけで
  よく、その後TDX Moduleに遷移したら必要に応じてコアのCPU命令
  を交えつつ処理が行われる
  - Reportの例で言えば、TDX Moduleは内部でSEAMOPS[SEAMREPORT]
    命令を発行する

- このインタフェースの内、TDX対応のHV向けのものを**SEAMCALL**、
  TD用のものを**TDCALL**と呼ぶ

TDX Moduleの配置場所

- TDX Module自体は、UEFI/BIOS経由で予約されたシステムメモリの一部である、**SEAM Range（SEAMRR）**上にロードされる
  - TDX Moduleのロードを行うP-SEAM Loaderも同じくSEAM Range上に配置される
- SEAM Rangeはこれ専用に用意されたTME-MK鍵によりAES-XTS暗号化が施されており、その内容が保護されるようになっている
- SEAM Rangeは、TDX Moduleのメンテナに直接確認した所、**SEAM Range専用のTME-MK鍵**で**暗号化される**と返答があった[2]
  - Demystified論文[1]や数々の公式文書からは特定できない情報

TDX Moduleの配置場所

- ちなみに、SEAMRRという表現はSEAM Rangeと同義のように使われるが、厳密にはSEAMRRはSEAM Rangeを設定する**レンジレジスタ**である
  - Intel公式ホワイトペーパー[3]でも同義のように使われている
- TDXに限らないレンジレジスタの例ではMTRRがあり、SGXにおけるPRMを指定するPRMRRもレンジレジスタの一種である
- 公式でも同義扱いされるくらいであるため、基本的に同じものと見なし、レジスタとして振る舞う部分では1つのレンジレジスタとして読み進めるくらいの感覚で良い

TDXにおける仮想化モード

- 前回のスライドでも説明した通り、VMXの2つのモードに加え、TDXでは**SEAM VMX root**モードと**SEAM VMX non-root**モードが追加されている
  - 前者はTDXの中でもTDX ModuleやP-SEAM Loaderのような最大の特権を要するコンポネントが動くモード、後者はTDが動くモード
- HVは従来通りVMX rootモードで動作するため、SEAMモードで動作するTDを直接操作する事はできず、代わりにTDX Moduleにその操作を依頼する必要がある
  - このためのHV用インタフェースが前述の**SEAMCALL**である

SEAMCALLとTDCALL

- SEAMCALLやTDCALL自体は、TDXで新たに登場した**CPU命令**として実装されている[4]

- SEAMCALLやTDCALLはさらに、具体的にどのような処理を行いたいのかのラベルとして機能する**リーフ関数**を受け取る
  - あるCPU命令とリーフ関数のセットがある場合、Inst[Leaf]のように表記する

- これはSGXにおいてもENCLS[EADD]のように散々使われているものである

SEAMCALLとTDCALL

- SEAMCALLのリーフ関数は、P-SEAM Loaderが受け取るもの以外は全て「**TDH.**」というプレフィックスがつく
  - 例：SEAMCALL[TDH.SYS.INIT]
  - 同様にTDCALLのリーフ関数は「**TDG.**」というプレフィックスがつく
- このリーフ関数は、CPU命令（SEAMCALL等）呼び出し時に対応する値を**アキュムレータ**（RAXレジスタ）に**格納**しておく事で**指定可能**である
- SEAMCALLが実行されると、論理プロセッサ（LP）はVMX rootからSEAM VMX rootに遷移しTDX Moduleでのコード実行を行う
  - TDX Moduleがタスクを完了させると、**SEAMRET**命令を実行してVMX rootのHVに制御を戻す

TDの動作

- TDもSEAM VMX non-rootという従来のVMXモードからは隔離されたモードで動作するが、標準的なVM同様、（VM外で動作していた）ユーザアプリケーションを**修正なしに動かす事ができる**

- 言い換えれば、既存ワークロードを基本的に修正する事無くTDに載せるだけでTDXというTEEの恩恵に預かれるという事である
  - ただしAttestation関係は除く

- これは、TEE（Enclave）の上で動作させるためにはもはや**苦痛なレベルの大幅な改造**を施さなければならなかったSGXのような**部分隔離型TEEに対する最も大きなメリットの1つ**と言える

TDの動作

- ただし、TD内のゲストカーネルに関しては、TDXを認識し、TDXに整合し、TDXのセキュリティ要件を満たすように改造しなければならない

- 具体的な変更の例としては以下の通り：
  - ゲスト内の仮想化例外（**#VE**）ハンドラを介した新しいTDX例外の管理
  - TDとTDX Module間の通信のための、ハイパーコール的なメカニズムの実装
  - I/O操作のためのメモリページをプライベートメモリから共有メモリに移行
  - Attestationのサポート

- これらの具体的な実装はOSの種類により異なる可能性がある
  - 例えばTDX対応ゲストLinuxカーネルの詳細実装はここに書かれている

TDの動作

- TDもVMである以上、その管理者であるTDX Moduleの力を借りなければならないケースは日常的に存在する
  - 通常VMがHVの力を借りるのと同様
- これを実現するためにTDX ModuleがTDに対して提供しているインタフェースが前述の**TDCALL**である
  - 前述の通り、TDCALLのリーフ関数はTDG.MR.RTMR.EXTENDのように**TDG.**というプレフィックスがつく
- 余談であるが、VM Entry, VM Exit, TD Entry, TD Exitのような表現はラッパー的な表現であり、そういう命令が存在するわけではない
  - あくまでVMLAUNCH命令、VMRESUME命令、及びコンテキストスイッチがさらに中核として存在している

TDXにおけるコンテキストスイッチ

- TDX Moduleは、HVとTDにそれぞれSEAMCALL、TDCALLというインタフェースを提供しているのは前述の通り

- かつ、HVとTDのやり取りは必ずTDX Moduleが仲介するため、**HVとTDで直接遷移が発生する事は無い**

- 結果として、TDXにおいては**HV↔TDX Module間**と**TD↔TDX Module間**の2つのコンテキストスイッチのみが存在する事になる

HV↔TDX Module間のコンテキストスイッチ

- ここからは、例としてSEAMCALL発行時のコンテキストスイッチの流れについて説明する

- TDの管理操作等のため、HVがSEAMCALLを発行すると、これに伴いHVのVMX rootモードからTDX ModuleのSEAM VMX rootモードに遷移する

- この時、HVとTDX Module間のコンテキストスイッチ（SEAMCALLやSEAMRET）でのみ使用される、**SEAM Transfer VMCS**というTDXで新たに登場したVMCSがロードされる
  - Intelの文書ではSEAM VMCSとも呼ばれている[3]

SEAM Transfer VMCS

- 本来、VMCSというのはゲストVM（non-root）とHV（root）とでの遷移時に使用されるものである

- よって、ホスト側HVとTDX Moduleという、いずれもnon-rootでない特権モード間の遷移で使われるのは、VMCS本来の用途から考えると不自然であるという見方もある
  - SEAM Transfer VMCSに文字通りVMをControlする要素は無いと言って良い

- これに関してはDemystified論文[1]も、単にHVとTDX Moduleとのコンテキストスイッチ手段以上の何物でもないと割り切った方が良いだろうと言及している

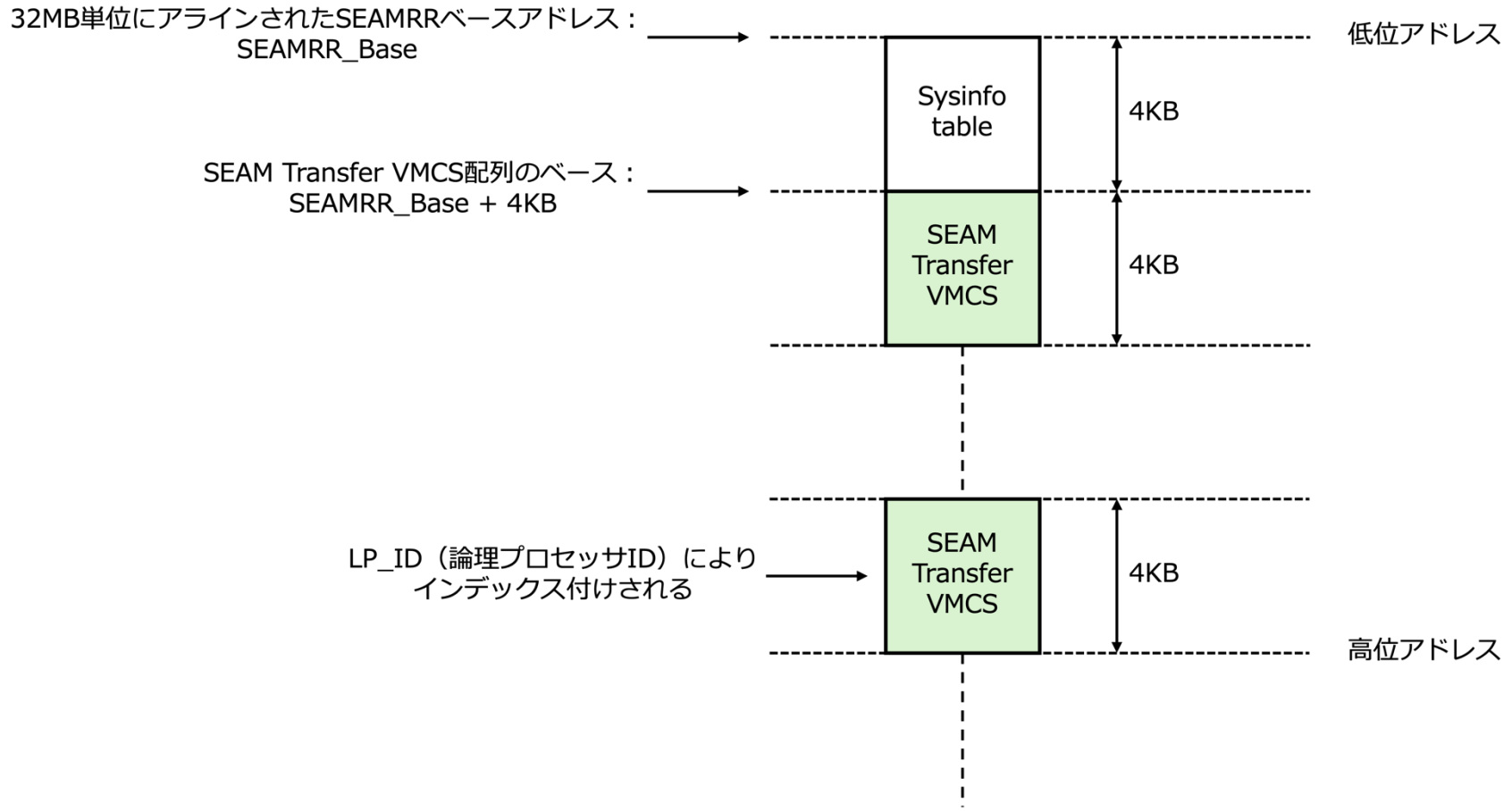
SEAM Transfer VMCS

- SEAM Transfer VMCSは、LP（論理プロセッサ）ごとにP-SEAM Loaderによってセットアップされる
  - 具体的には、SEAM Rangeのベースアドレス（先頭）である**SEAMRR_Base**から始まる Module_Range（TDX Module用物理メモリ範囲）内において、**Sysinfo_Table**という構造の真後ろに配置される
  - TDX Moduleの具体的なメモリレイアウトはTDX Module編で解説

- このVMCSはSEAM Range内に配置されるわけであるため、SEAM VMX rootからのみアクセス可能かつSEAM Range専用のTME-MK鍵により暗号化される非常に強力な隔離保護下で管理される
  - 暗号化されているので、SEV-ES以前の初代SEVのように、コンテキストを格納するこのVMCS（SEVではVMCB）が平文で野晒しになる事は無い

SEAM Transfer VMCS

- SEAM Transfer VMCSのLPごとの物理メモリレイアウト図を以下に掲載する（[1]を参考に作成）



SEAM Transfer VMCS

- SEAM Transfer VMCSは、Sysinfo_Tableの次、つまりはSEAMRR_Base+4kBから始まる

- かつLPごとに存在するため、全体としては**LPごとのSEAM Transfer VMCS領域からなる配列**として存在する
  - このVMCS1つにつき4kBページが割り当てられており、各SEAM Transfer VMCSはLP_IDというLPの識別子によりインデックス付けされる

- 以下、このHVとTDX Module間のコンテキストスイッチに関して単にVMCSと呼んだ場合、SEAM Transfer VMCSを指すものとする

SEAM Transfer VMCS

- LPは、自身に割り当てられたVMCSを使用しなければならない
  - よって、SEAMCALLが発行されると、プロセッサは現在のLP_IDに基づいてVMCSアドレスを検索する
  - 具体的にはSEAMRR_Base + 4kB + (LP_ID * 4kB)の式によりアドレスが決定される

- 肝心のTDX Moduleのステートは、このVMCSの**Host State Area**という領域に格納される[1]
  - P-SEAM LoaderにおけるVMCSのセットアップ用ソースコードを見ると、このVMCSが具体的に何を含むかを間接的に推測できる[5]

SEAM Transfer VMCS

- このHost State Areaには、例えばホストRIPレジスタのステートとして、TDX Moduleのtdx_seamcall_entry_pointというものが格納される
  - RIP：命令ポインタレジスタ

- また、CR3レジスタ（制御レジスタ3）のステートとしては、TDX ModuleのPML4のベース物理アドレスが格納される
  - CR3レジスタ：現在有効なページテーブルの情報を格納するレジスタ。詳細はこちらを参照
  - PML4：ページテーブル構造の階層化により構造全体のサイズの削減を目的とした、4-レベルページテーブル（PT, PD, PDPT, PML4からなる）の最上位に位置するページテーブル構造の事

SEAM Transfer VMCS

- 他にも、このAreaのFS_BASEステートにはSysinfo_Tableの仮想アドレスが、GS_BASEステートには各LPごとのデータ領域（詳細はTDX Module編で説明）が設定される
  - FSやGSというのはセグメントレジスタの一種であり、FS_BASEやGS_BASEはFSやGSが参照するベースアドレスを格納するモデル固有レジスタ（MSR）である
- これらのステート情報をVMCSに格納した状態でTDX Moduleに遷移すると、これらの情報が**実際にプロセッサにロード**される

SEAM Transfer VMCS

- 同時に、この遷移に伴い、**メモリ管理ユニット**（MMU）は**TDX Moduleの仮想アドレス空間で動作**するように**切り替わる**

- その後、 TDX Moduleは**SEAMCALL処理を開始**し、リーフ関数で指定された処理を行う対応するインタフェース関数への**ディスパッチ**（対応する関数の実行）を行う

- 今回はTDX Moduleに対するSEAMCALLの場合を例に取ったが、それに限らずHVとTDX Moduleとでのコンテキストスイッチは、このようにSEAM Transfer VMCSを多分に活用しながら行われている

TD↔TDX Module間のコンテキストスイッチ

- 前提知識として、従来のVMにおいてはHVがVM Exitを処理し、これは**従来のVMCS**により制御される
  - MCSはそれぞれ1つのvCPUに関連付けられ、次のVMRESUME（VM再開命令）時にゲストの実行を復元する時のためにCPUステートを保存しり
- ちなみにLPとvCPUは、LPに対してvCPUをスケジューリングするという関係性を持つ
  - 例えばある1つのLPをvCPUに使用するのであれば、そのLPに対し1つまたは複数のvCPUを割り当てるイメージ

TD↔TDX Module間のコンテキストスイッチ

- しかし、この従来型VMCSはHVから見えてしまうため、このままでは**ホスト側にvCPUステートが漏洩**してしまう事になる
  - TDXはvCPUステートの保護も行わなければならないため、これは当然に許される事ではない
  - 実際にこのVMCS相当のデータ構造（VMCB）を暗号化していなかったがために問題となったのが、SEV-ES導入前の初代AMD SEVである

- これを対策するため、同期的TDCALL、同期的TD Exit、または非同期TD脱出（**Asynchronous TD Exit**）といったあらゆる場合にてTDX Moduleが必ず仲介（トラップ）する
  - このAsynchronous TD Exitは、SGXにもAsynchronous Enclave Exit（AEX）という似た仕様が存在している

TD↔TDX Module間のコンテキストスイッチ

- TDとTDX Module間の同期・非同期遷移のいずれの場合も、TDXにおいては**TD Transfer VMCS**という専用のVMCSが使用される
  - Intel公式文書[3]ではTD VMCSとも呼ばれている

- こちらはSEAM Transfer VMCSとは異なり、VMであるTDとHV的な管理を行うTDX Moduleとの遷移で使用されるので、従来の定義通りのVMCSであると言える
  - このVMCSが具体的に何を格納しているかは、TDX Moduleにおける TD Transfer VMCS初期化用ソースコード[10]を見る事で分かる

TD↔TDX Module間のコンテキストスイッチ

- TD Transfer VMCSは、TDのvCPUが作成され、それがそのTDの**TDVPS**に格納される際にセットアップされる
  - TDVPS : Trust Domain Virtual Processor State

- TDVPSは、そのTDの各vCPUに対して1つずつ割り当てられる、vCPUについての制御構造であり、TD Transfer VMCSを内包している
  - 詳細はTDX Module・メタデータ編で解説

TD↔TDX Module間のコンテキストスイッチ

- TDVPSはそのTDのメモリを暗号化するTME-MK鍵で同じく暗号化されるため、HVがTD Transfer VMCSにアクセス・盗聴する事は不可能である

- TDがTDCALLを呼び出すか、またはTD Exitをトリガーすると、LPはTD Transfer VMCSに格納されたTDX Moduleのステートをロードし、コンテキストスイッチを実施する

TDXと従来型HVの連携

- ポートI/O、HLT命令、CPUID命令といった特定の操作は、TDにおいても**HVによるエミュレートが必須**である
  - このような操作がTD内で行われると、TDは救援（エミュレート）を受けるべくTD Exitを行う

- すると一旦TDX Moduleに制御が渡るが、上記のような操作はTDX Moduleもエミュレートできないため、さらに外側にいるVMX rootのHVの助けを借りる必要がある

TDXと従来型HVの連携

- 従来型VMであればHVが直接vCPUステートとメモリ全体を見てこの処理を行っていたが、TDXにおいては当然そのような事は許されない

- TDXでは、HVへの情報公開が最低限で済むよう、新たに**#VE**という例外が導入された
  - VE: Virtualization Exception（**仮想化例外**）

TDXと従来型HVの連携

- 以下のように#VEを使用する事で、最低限の情報の公開のもとにHVによるエミュレートを実現できる：
  1. TDは、特定の操作のエミュレートのためにTD Exitリクエストを発出する
  2. TDX Moduleがそれをトラップし、#VEをTDに注入する
  3. TDのゲストカーネルに実装されているVMハンドラは、必要なパラメータの一式を準備する
  4. VEハンドラは、再度TDCALLを発行しHVに制御を移す
  5. TDX Moduleは、受け取ったパラメータに基づいてHVに処理を委託し、必要な処理を実施してもらう

- ちなみに、SEV-ES以降のAMD SEVでは#VC例外というものがTDXの#VE相当として用意されている
  - TDXの#VEとは異なりCPUから発出される

TDXにおけるEPT

- 前回スライドで説明した通り、従来型VMではHVがEPTを使用してGPAからHPAへのアドレス変換を管理する

- しかしTDXではHVを信頼不可能とするため、アドレス変換を安全に行うためにSEPTと共有EPTの2種類のEPTを用意している

TDXにおけるEPT

- SEPTは、TDのプライベートメモリのアドレス変換に使用され、そのTDのメモリ暗号化鍵（TME-MK鍵）によって保護される

- このSEPTを参照する際には、まずTDのメタデータを格納する構造である**TDCS**内の、**SEPTルートページ**というページを参照する
  - TDCS: Trust Domain Control Structure

- SEPTルートページはSEPT本体への参照として機能するため、これを介する事でSEPT本体を参照する事ができる

TDXにおけるEPT

- 一方で共有EPTは、仮想化I/Oのケースのように、TDがHVと明示的に共有するようなメモリ（つまりは共有メモリ）のアドレス変換に使用される
  - 共有EPTはHVの制御下に存在する

- TDのゲストカーネルは、GPAの**Sharedビット**を1にする事でそのページを共有メモリ扱いにできる
  - Sharedビットは、有効なGPA幅の中での最上位ビットがその役割を負う[7]
  - 共有メモリのページはTDの秘密鍵では暗号化されない

TDXにおけるEPT

- AMD SEVでは、似たような仕組みとしてCビットというものが存在する
  - TDXのSharedビットとは逆に、1にする事でそのメモリページをプライベートメモリに指定する事ができる

- SEPTを利用しながらの物理メモリ管理に関する詳細な説明はTDX Module編のスライドで実施

まとめ

- Intel TDXにおける仮想化周りがどのようになっているのかについて詳細に学習した

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] “Questions Regarding SEAMRR Encryption”, GitHub, https://github.com/intel/confidential-computing.tdx.tdx-module/issues/25

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

[4] “Intel TDX Deep Dive”, Dr. Benny Fuhry, https://archive.fosdem.org/2024/events/attachments/fosdem-2024-2608-intel-tdx-deep-dive/slides/22764/240204_Intel_TDX_Deep_Dive_KrjIuZM.pdf

[5] “seam_vmcs_setup.c”, GitHub, https://github.com/intel/confidential-computing.tdx.tdx-loader/blob/tdx-loader-1.5/p-seam-loader/src/common/data_structures/seam_vmcs_setup.c#L34

[6] “td_vmcs_init.c”, GitHub, https://github.com/intel/confidential-computing.tdx.tdx-module/blob/tdx_2.0/src/common/data_structures/td_vmcs_init.c#L419

[7] "Intel® Trust Domain CPU Architectural Extensions", Intel, https://cdrdv2.intel.com/v1/dl/getContent/733582