TDX.FAIL

本セクションの目標

- 単にTDXの機能を使うだけでは秘密を保護しきる事ができない
  各種要素について説明する

- それぞれはその解決のために難しい設計や実装が必要となるため、
  セキュリティキャンプ本番における応用編の課題となり得る

ストレージの暗号化

ストレージの暗号化

- TDXを始めとしたCVMは、あくまでもそのVMのメモリを暗号化する機能しか持たない

- つまり、例えば単にCanonicalの公開している手順に従いTDを構築しただけでは、そのTDにアタッチされている仮想ディスクの内容を暗号化する事はできない

- これは秘密情報を一切不揮発性ストレージで保管できない事を意味するため、このままでは非常に不便である

ストレージの暗号化

- しかし、CVMはゲストOSを丸ごと内包するため、OSが提供するフルディスク暗号化（FDE）を用いて仮想ディスクを暗号化する事ができる
  - FDE：Full-Disk Encryption
  - まともなクラウドが提供するCVMであれば基本的にFDEは適用されている
- Linuxにおいては、LUKSという機能を用いるとFDEを実現できる

応用課題①

- LUKSを用いてTDにアタッチされた仮想ディスクをフルディスク暗号化せよ。

ゲストメモリの共有化

ゲストメモリの共有化

- 以前のスライドでは、TDのゲストカーネルからゲストメモリのSharedビットを1にする事で、そのページを共有メモリにできると説明した

- これはもし特にランタイムワークロードやゲスト内構成を正常にRAで検査できていない場合、sudo付きで実行された不正なワークロードが勝手に共有メモリを作れてしまう可能性がある事になる
  - だからこそ、期待する構成に対してリファレンス測定値を取り、RAでそれを確認する事が重要である

応用課題②

- TDゲスト内から特定のメモリページのSharedビットを1にせよ。
  ただし、root権限を行使しても良いものとする。

破綻しているTDXのRA

破綻しているTDXのRA

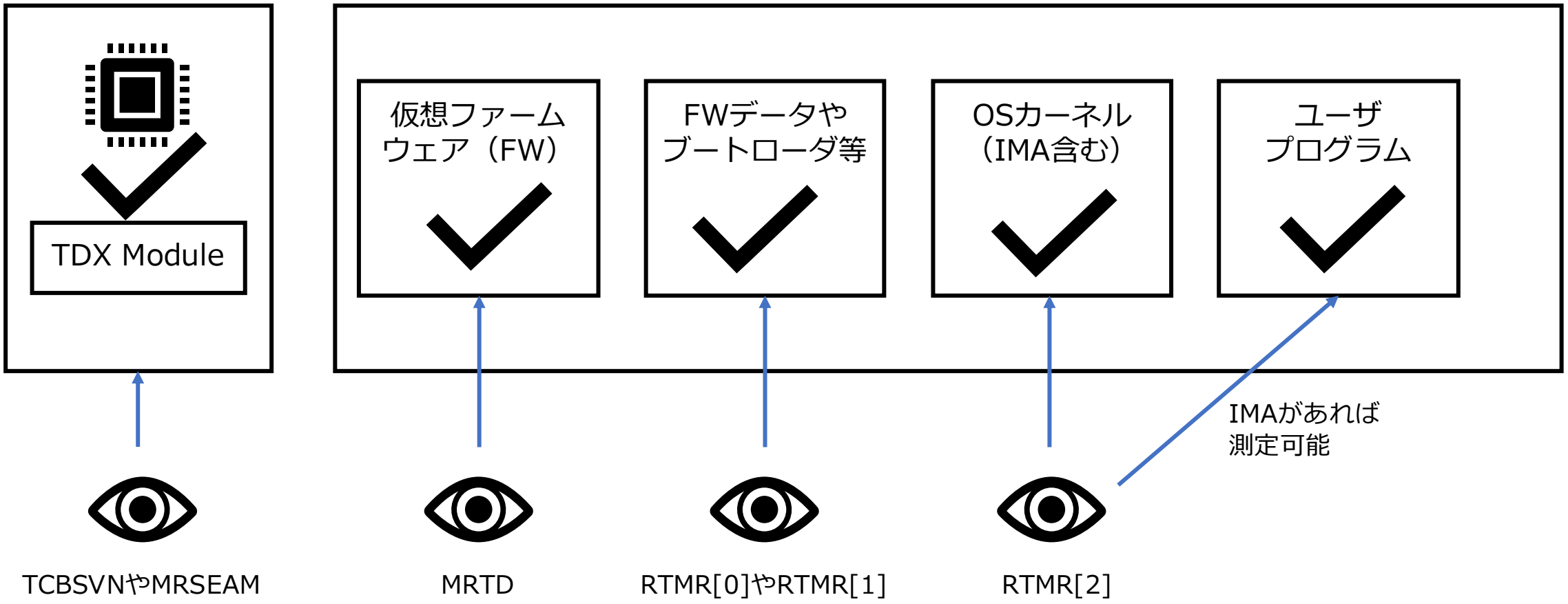
- TDXのベアメタルマシンでRAをしてみた所、なんと**IMAによる測定結果がRTMR[2]に反映されていない**事が判明した

- IMAはLinuxカーネルの機能で、カーネル以降のアッパーレイヤを測定・記録してくれる機能
  - IMA：Integrity Measurement Architecture
  - 通常は、TPMのPCRという測定レジスタのバンク10にIMAによる測定値がExtendされる

- TDXでは、代わりにRTMR[2]にExtendされると**公式が明言している**

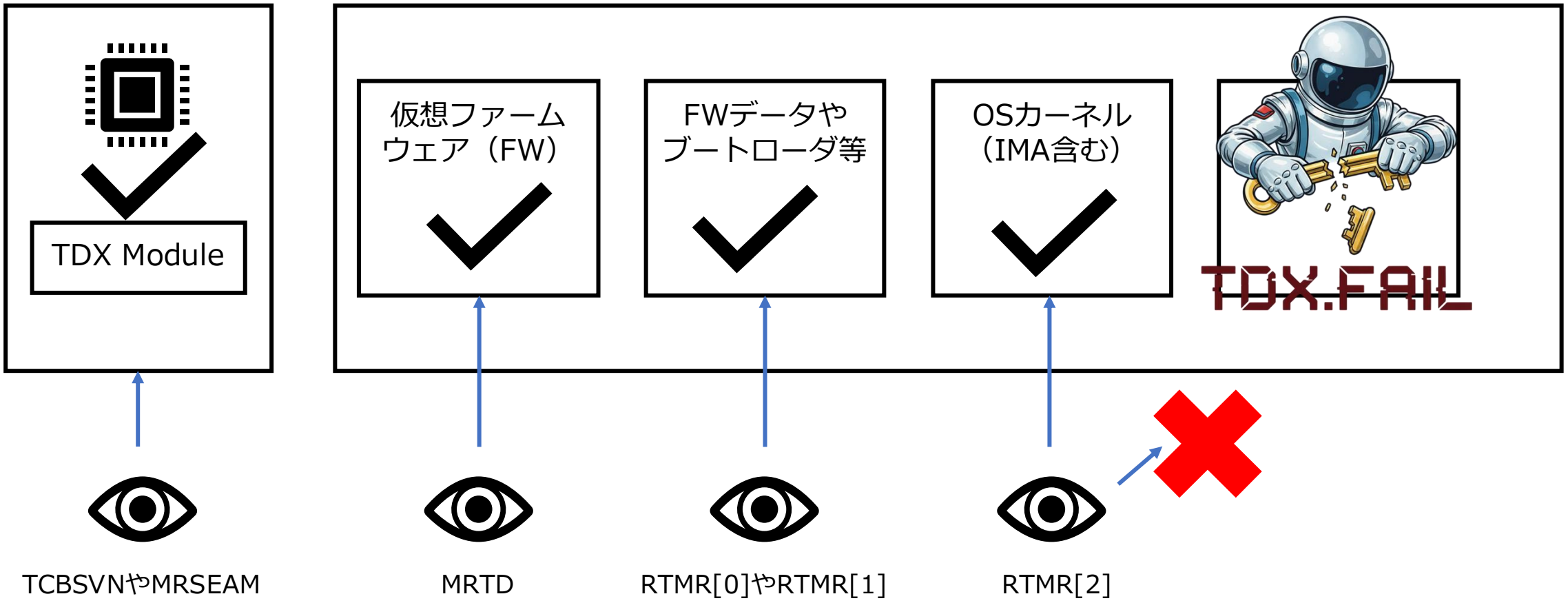
破綻しているTDXのRA

- ちゃんと実装されていれば、以下のようにTDは全面がしっかりと厳格に記録され、不審者は検知できるようになっている



破綻しているTDXのRA

- が、IMAがRTMR[2]で測定できていないのでこうなる



破綻しているTDXのRA

- 公式にもそれらしきIssue及び修正PRが出ており、何ならIntel社員からPRは出ているが、2026/7/6現在も一向にマージされる気配は無い

extend: align with Linux upstream extend interface #473

Open fengyuleidian0615 wants to merge 1 commit into intel:main from fengyuleidian0615:my-align-extend-with-upstream-de

Conversation 0 Commits 1 Checks 0 Files changed 1

fengyuleidian0615 commented on Dec 2, 2025 • edited

With merge of unified extend interface in Linux 6.16, now switch to standard interface, and abandon previous ioctl.

Linux/Documentation/ABI/testing/sysfs-devices-virtual-misc-tdx_guest

The inconsistent interface issue was orignally reported here: #472

extend: align with Linux upstream extend interface 435fda1

This branch has conflicts that must be resolved

Changes can be cleanly merged.

Resolve conflicts

Fan Du

fengyuleidian0615

Follow

6 followers · 3 following

Intel

Beijing

破綻しているTDXのRA

- …と思いきや、このPRが放置されているだけで、Linux 6.16及び対応するツール側の更新でこのPRが挙げていた問題は解決された
  - ゲストから利用可能なAttestation用の公式ツールtdx-guestが、正しいパス（/sys/devices/virtual/misc/tdx_guest/measurements/rtmr2:sha384）にExtendするように修正された

- ちなみに上記以外のExtend先候補も過去の紆余曲折の中乱立している（正しいのは上記及び以下太字のsymlinkのみ）：
  - /sys/kernel/config/tsm/rtmrs
  - **/sys/class/misc/tdx_guest/measurements/rtmr%d:sha384**
  - /sys/devices/virtual/misc/tdx_guest/mr/rtmr%d:sha384
  - /sys/devices/virtual/misc/tdx_guest/rtmr%d:sha384
  - /sys/class/misc/tdx_guest/mr/rtmr%d:sha384

TDX.FAIL

破綻しているTDXのRA

- が、これを直してもまだRTMR[2]にExtendされない！
  これは一体何故だろうか！？

破綻しているTDXのRA

- 何故ならば、前述のIntelの公式文書での説明とは裏腹に、そもそもIMAがRTMR[2]にExtendする機能自体がLinuxカーネルおじさん達に拒否されていたためである
  - 先程の6.16での修正は、あくまでもユーザのツールレベルの話で、それとは別にそもそもカーネルが根本的な機能に対応していないという大どんでん返し

- 議論と拒否の経緯は以下のリンク先から見られる：
  - https://www.mail-archive.com/linux-integrity@vger.kernel.org/msg04008.html
  - https://www.mail-archive.com/linux-integrity@vger.kernel.org/msg01077.html

この辺の経緯等はチューターの伊藤さん調べ

破綻しているTDXのRA

- つまり、TDXは現状**ランタイムコンテンツ**を**そのままでは測定できない**という**トンデモCVM**である事が判明した
  - 公式により引導を渡され既に消滅したリポジトリのハードフォークの謎のアーカイブの中に存在するコードには**ima_extend_rtmr**というExtend機能があるが、古いカーネル向けで現行には使えない
  - 何なら公式文書ではima_**rtmr_extend**と書いてあったので、定義したAPIすらもまともに参照できていない

- 現状でまともにランタイムコンテンツを測定できているのは、TD-PartitioningによりvTPMをゲスト内でさらに隔離実行しているAzure TDX VMだけであると言えよう
  - GCPのは普通にホスト側TPMに測定を投げるらしい（Acompany社員談）。これではTDXの脅威モデル的にはNG

CVMのRAの難しさ

- 仮にこれが解決されたとて、**CVMのRAというのは非常に難しい**

- 例えば以下をどのように測定するか？
  - ルートファイルシステム（仮想ディスクの内容）
  - Pythonで対話式にインラインで入力されたスクリプトの実行
  - bashコマンドライン上で定義・実行が完結するようなコマンドの実行

CVMのRAの難しさ

- ディスクの内容であれば、例えばdm-verityのようなディスク完全性保証ツールと組み合わせ、RTMRにバインドさせる方法がある
  - が、dm-verityは読み取り専用であるため、これだけというわけにもいかない
- PythonやBashのコマンドライン完結実行に関しては、そのコマンドのIMAのファイル読み取り検出にすら引っかからないため、Executability Check等を用いる必要がある
- その他、変なものがランタイム中にダウンロードされて実行されないように気をつける（例：curl, apt）等、**IMAで検知できたとしても注意深く見なければならない要素も多い**

CVMのRAの難しさ

- そもそもIMAがExtendするPCR10は、ロードされるコンポネントの順序が起動の度に異なる上に随時追加されるため、RP/Verifierが毎回測定ハッシュ値を**期待するコンポネントで再計算**する必要がある
  - IMAログというものがカーネルから取得できる

- が、IMAは例えば「ファイル全部」「共有ライブラリ全部」の「読み取り」「実行」みたいな粒度でしか指定できないため、**全然測定できない**か**測定対象量が爆発するか**の**両極端**である

- 理想的には全部測る事だが、**ゲストOS上の全コンポネントに対する夥しい量の再計算を誰がやれるというのか？**（修辞疑問文）

CVMのRAの難しさ

- つまり、TDXに限らずCVMのランタイムRAは現実的には実用上厳しいものがあると言わざるを得ない

- Confidential Container（CoCo）という技術では、初期メモリ配置コンテンツのハッシュ値（例：MRTD）で縛れるコンテンツが定めるポリシで上流の動きを全て制御する設計となっている
  - ランタイム中もポリシ外の未検出での動きはできず、そのポリシの強制はMRTD等レベルで行われるため、1つの解であると言える

応用課題③

- [極悪難易度]IMAの測定結果がRTMR[2]にExtendされるように改造せよ。

- 例えば以下のようなコンテンツをRAで縛るための実装や構成を実施せよ。
  - ルートファイルシステム（仮想ディスクの内容）
  - Pythonで対話式にインラインで入力されたスクリプトの実行
  - bashコマンドライン上で定義・実行が完結するようなコマンドの実行

- [超極悪難易度]IMAがRTMR[2]にExtendしないというのであれば、TD-PartitioningとOpenHCLを用いてAzureと同様の基盤を構築せよ。

GPU Attestation

GPU Attestation

- キャンプ本番で使用するマシンには、**NVIDIA Confidential Computing（NCC）に対応**している**H100を搭載**している（予定）
- つまり、**GPU TEEも織り交ぜながらのTEEアプリケーション開発が可能**である
- しかし、GPUを用いるのであれば**GPUに対するAttestationをしなければならない**

GPU Attestation

- NCCを併用する場合、まず**CVMからNCCをAttestation**し、**そのCVMをRP/VerifierがCVMに対するRAで検証**する
  - NCCにおけるAttestation Reportは一般にEvidenceと呼ばれる

- これにより、**RP/Verifierが正しいと確信したCVMが正しくNCCを Attestationした**という事を**確信**できるようになる

- しかし、これも何をRP/Verifierに返すか（NCCのEvidenceも RP/Verifierに一応改めて検査させるか）等、**設計の幅は意外に広い**

GPU Attestation

- AzureのSEV-SNP + NCCインスタンス向けには、このような連鎖RAを行う**Humane-RAChain-NS**を開発・公開している
  - https://github.com/acompany-develop/Humane-RAChain-NS

- 上記リポジトリの実装では、IMAでPCR10に全部Extendさせる事もできるが、特に分かりやすくみたいものは便宜的にPCR23にも特定のコンテンツをExtendしている：
  - NCC Evidence
  - 利用しようとしている直接のワークロードのコード等
  - より厳密に見るならばIMAを厳しく設定しPCR10、カジュアル重視ならPCR23を見る、というイメージ

応用課題④

- Humane-RAChain-NSのTDX版とも言える**Humane-RAChain-NT**を実装せよ。

TDX.FAIL