【TERM 最終発表】
RDMAを用いた，遠隔ベアメタルマシン デバッグのための仮想アドレス空間の復元

Arch B3 tatsu
親: macchanさん,soraさん

背景

- OSデバッグやメモリフォレンジックをする際,
  物理マシン・仮想マシンを解析
  - 特に，カーネルパニック時の解析にはVMを利用
  - XenやQEMU＋KVMなど
- 関連ソフトウェア: libvmi, google/rekallなど

課題

- 物理マシンに対して，カーネルパニック時におけるメモリ解析手段が存在しない
- OSが提供していた機能が使えなくなるため
  - Memory Management Unit
  - カーネルのAPI
  - カーネルシンボル

目的

- 遠隔ベアメタルマシンの物理メモリの監視アーキテクチャを確立
  - 外部リソースで動作
- **カーネルパニック時に特定のプロセスの仮想アドレス空間を復元**

ここに図をかく

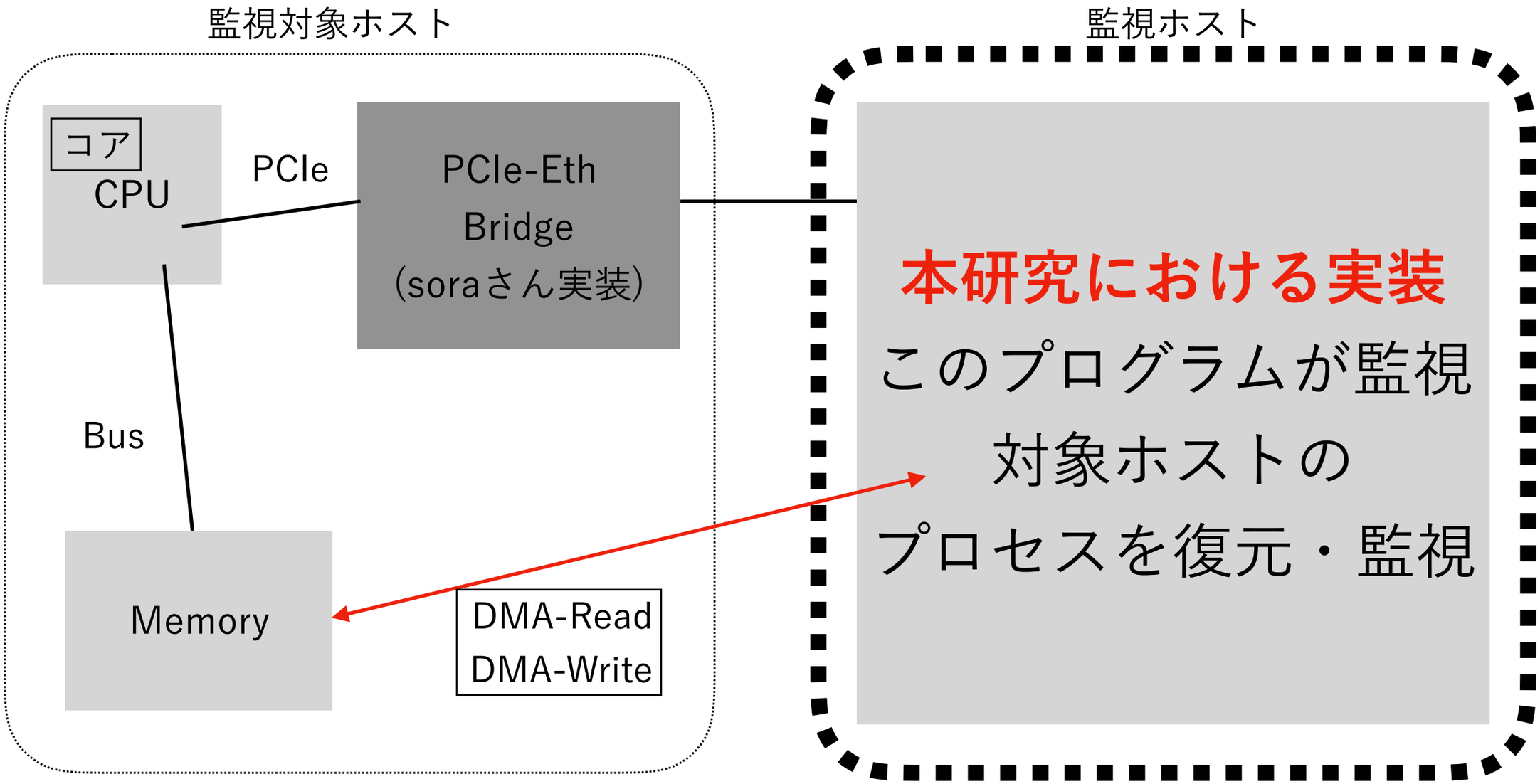
アプローチ

- ページウォークのロジックを解析
  - Linuxでは，**CR3**(レジスタ)の値をもとに，ページウォークをし仮想アドレスを物理アドレスに変換
    - 実際はCPUキャッシュがある
- 全ての情報を本研究のプログラムで保持
- ロジックを逆算し仮想アドレス空間を復元

実験環境

- 監視プロセスが動くホスト
  - x86-64 Linux
- 監視対象ホスト
  - x86-64 Linux

システム概要



PCIe-Eth Bridgeのデータ構造

システム概要

- PCIe-Eth Bridge(soraさん実装)
  - PCI MessageとEthernetパケットを変換するFPGAデバイス
  - プログラム中から物理アドレスを指定しデバイスを呼び出す
    - 値を4Byteずつ取得
- PCIe-Eth Bridgeを監視対象ホストに設置
- PCIe Eth Bridgeの手続きを大量に呼び出す
  - 連続した領域の値を取得

実装

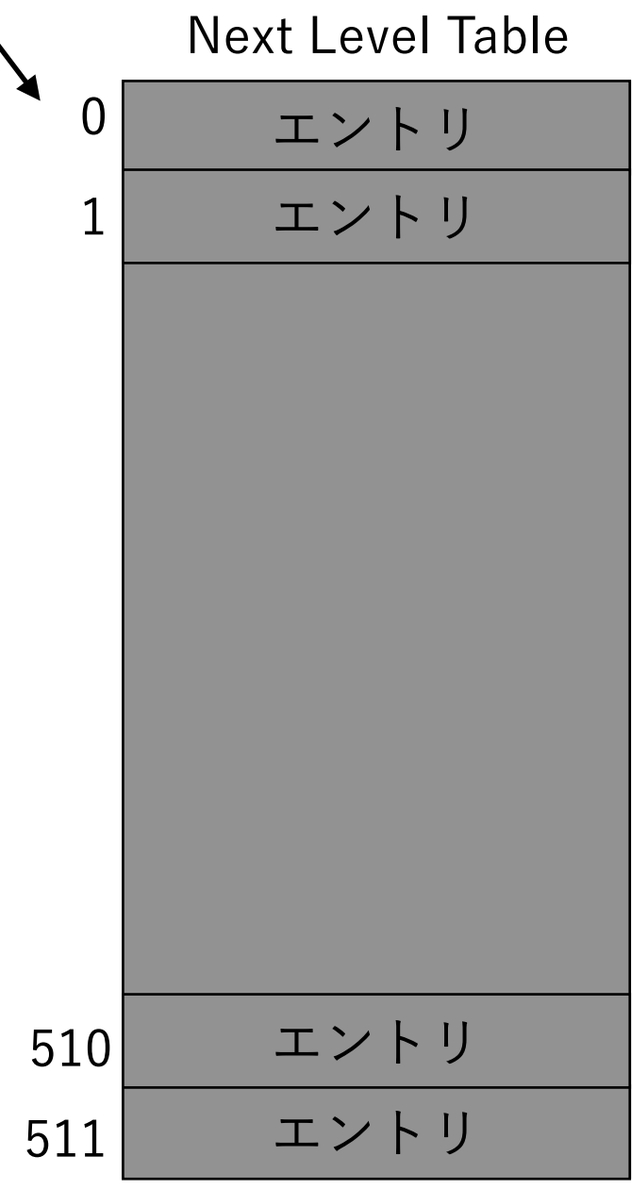
- CR3の値を**監視対象ホストから通知**
  - CR3はレジスタである
    - 現在の値をメモリから参照できない
- 4階層のページウォーク
- ソース: https://github.com/dooooooooingggggg/dmaLinux/blob/master/src/dmaLinux.c

実装

- Pは，エントリの先に物理アドレスが設定されているかどうかのフラグ

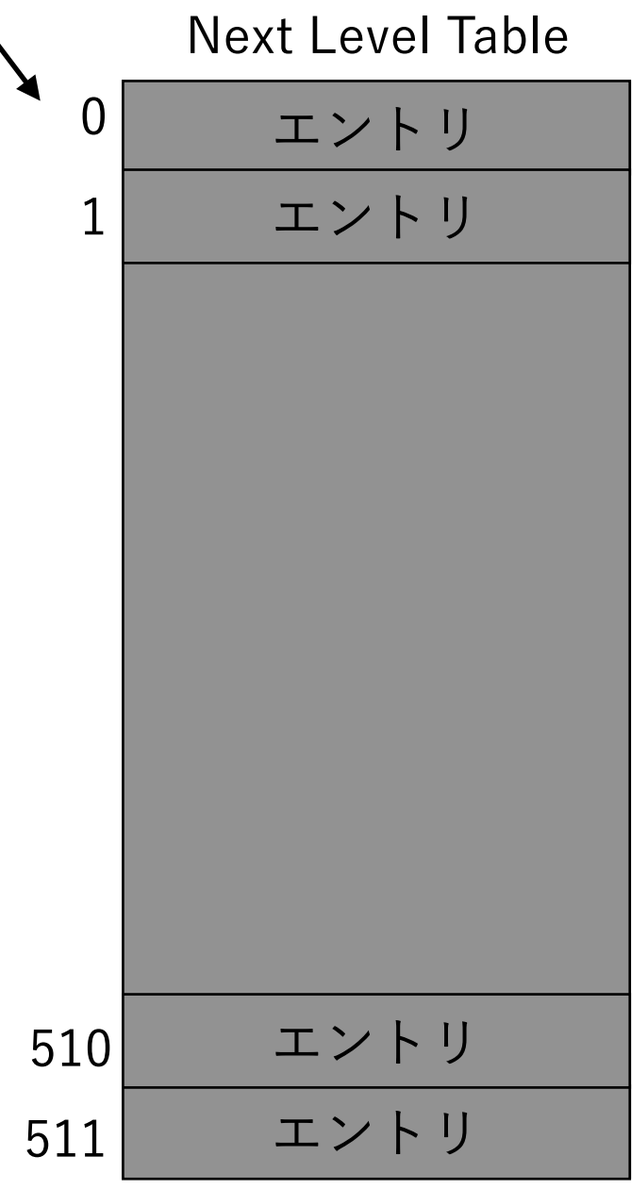
- 最下層(Page Table)の39-12bitには，物理アドレスが格納されている

  - 4KB境界



実装

- 通知された値を起点に，4階層のテーブルを取得
  - 愚直にやると，512 ** 4個のエントリを読み込む必要
  - エントリごとに，Pフラグが0の場合は次の階層からスキップ

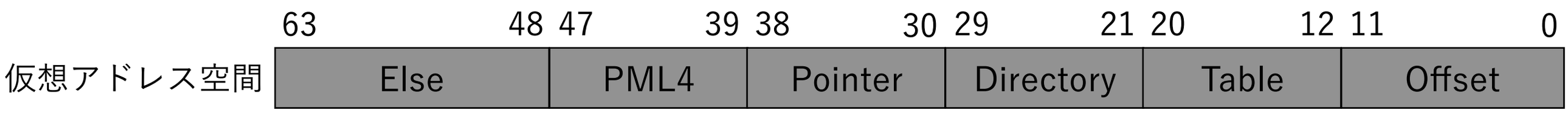


実装

- 4階層のテーブルをたどり物理アドレスおよび，値を取得

- 以下の処理に通すことで仮想アドレスの復元が完了

```
(pml4Index << 39)+ (pdpIndex << 30) +
(pdIndex << 21) + (ptIndex << 12) +
offset;
```



評価

- **カーネルパニック時の対象プロセスの仮想アドレスの復元の可否**
  - カーネルパニックがおきた時間の特定
    - 現在の時間を変数に保持するプロセスを対象
  - 意図的にカーネルパニックを起こす

評価

- 監視するプロセス
- https://github.com/dooooooooingggggg/getcr3/blob/master/user.c
  - CR3の値をカーネルモジュール経由でprintf()
    - macchanさん実装のカーネルモジュールを拡張したものを使用
- 現在のUNIX TIMEをprintf()

評価

- 出力結果

```
# 0x7fff65099000(phys[0x2697c9000]) : 0x0
# 0x7fff65099008(phys[0x2697c9008]) : 0x0
# 0x7fff65099010(phys[0x2697c9010]) : 0x0
# 0x7fff65099018(phys[0x2697c9018]) : 0x0
# 0x7fff65099020(phys[0x2697c9020]) : 0x0
# 0x7fff6509aa8c(phys[0x2697e5a8c]) : 0x5c4970de00000028
# 0x7fff6509aa94(phys[0x2697e5a94]) : 0x6509aa9000000000
# 0x7fff6509aa9c(phys[0x2697e5a9c]) : 0x8005003300007fff
# 0x7fff6509aaa4(phys[0x2697e5aa4]) : 0x75cf3b0000000000
# 0x7fff6509aaac(phys[0x2697e5aac]) : 0x309c3230ffff8802
# 0x7fff6509b000(phys[0x269cba000]) : 0x0
# 0x7fff6509b008(phys[0x269cba008]) : 0x0
# 0x7fff6509b010(phys[0x269cba010]) : 0x0
# 0x7fff6509b018(phys[0x269cba018]) : 0x0
# 0x7fff6509b020(phys[0x269cba020]) : 0x0
```

- **0x7fff6509aa90**に変数の値が見える
- **5c4970de**(= **1548316894** (UNIX TIME))にカーネルパニックが発生

結論・今後の展望(卒論に向けて)

- CR3が通知された状態において，仮想アドレス空間の復元が可能なことが実証
  - カーネル空間にあるtask_structをメモリのみから探す
    - CR3の値はtask_struct->mm_struct->pgdに格納
    - CR3の通知なしで全プロセスの復元が可能になる
- 全てのテーブルを検査できるようにデバイスを修正
  - 現在はindexやoffset自分で指定し入力している