【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

目次

背景

- OSデバッグやセキュリティフォレンジックするために物理・仮想メモリの解析が利用される
  - 通常OSのメモリ解析には仮想マシン(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

  - 内部リソース・ロジックを使用するため

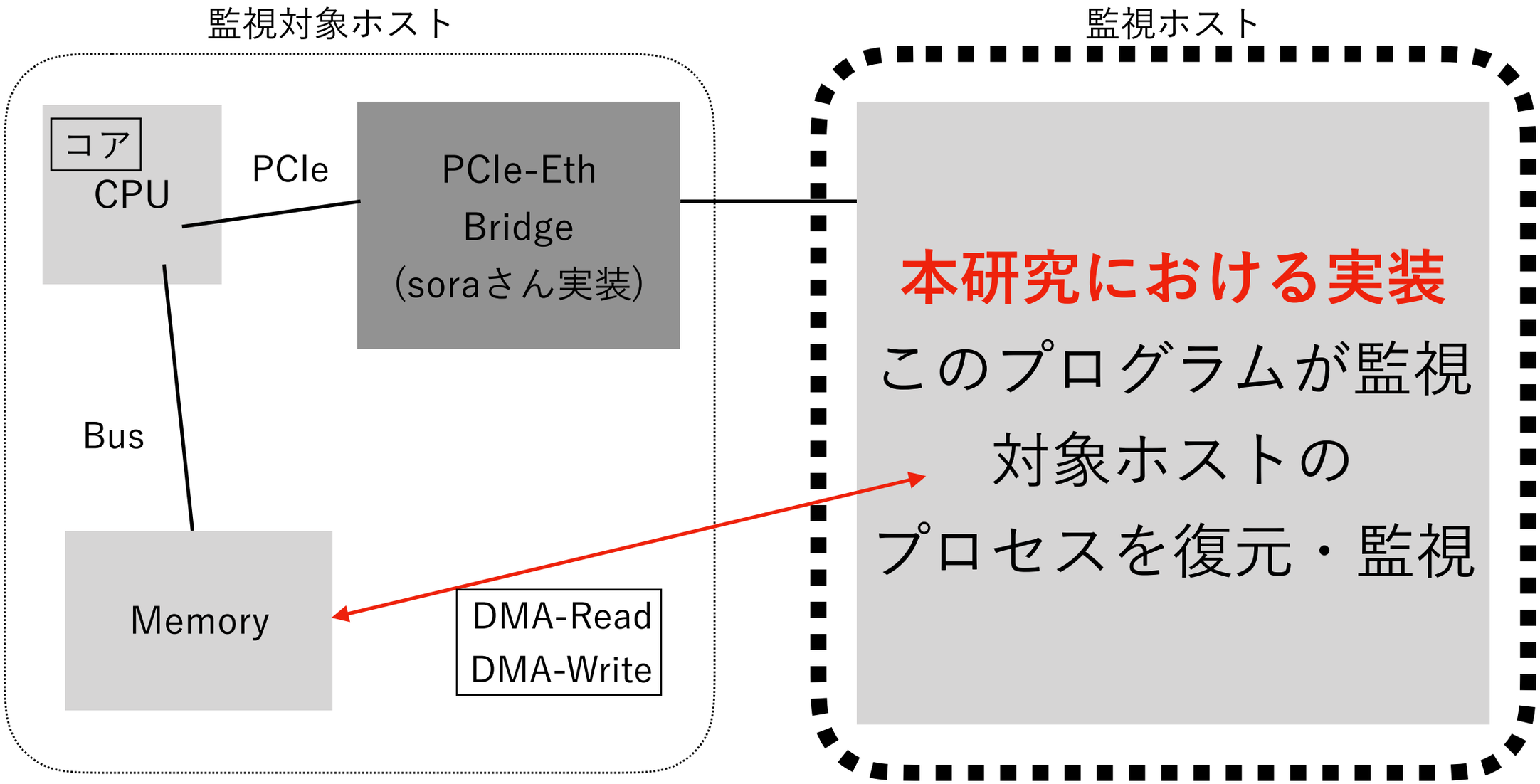
目的

- 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**

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

アプローチ



PCIe-Eth Bridgeのデータ構造

アプローチ

- 物理アドレスを指定することで，値を4Byteずつ取ることができるFPGAデバイス(soraさん実装)

- PCIe Eth Bridgeの手続きを大量に呼び出すことにより，連続した領域の値を取得

x86-64 LinuxのMMUの仕組み

- セグメント機構
  - 論理アドレスからリニアアドレスを算出する機構
- ページング機構
  - リニアアドレスから物理アドレスを算出する機構

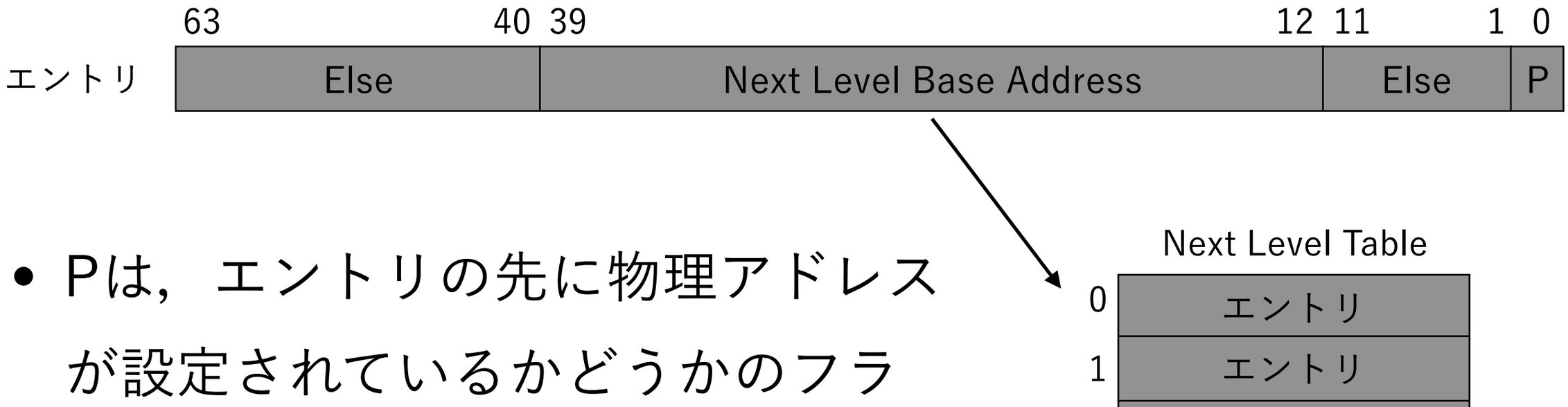
x86-64 LinuxのMMUの仕組み

- セグメント機構・ページング機構はプロセッサに用意
  - Linuxではページング機構のみ採用
- プロセッサの制約上，セグメント機構は無効にできない
  - セグメント機構はフラットになるように設定
- プロセスでは，アドレスの参照に**仮想アドレス**(リニアアドレス)を使用
  - ページング機構を通し，物理アドレスを算出

x86-64 LinuxのMMUの仕組み

- Page Map Level 4
    - Page Directory Pointer
        - Page Directory
            - Page Table
- 4段階の変換テーブルが存在
- それぞれ，512個のエントリ・1エントリのサイズは8 Byte
- テーブル全体のサイズは，4KB (8 Byte * 512)

x86-64 LinuxのMMUの仕組み



- 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である

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

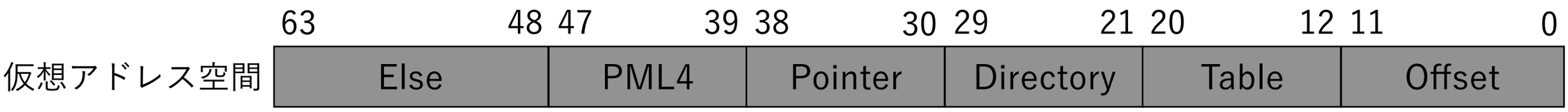
  - 4KB境界

x86-64 LinuxのMMUの仕組み

- プロセスごとに4階層のテーブルを保持・アドレス空間の独立を実現

- 起点となる，PML4のベースアドレスは**CR3レジスタ**に保持.

x86-64 LinuxのMMUの仕組み

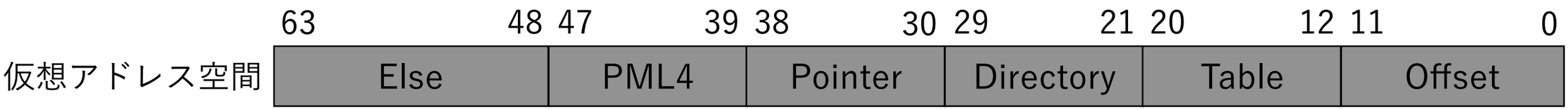


- PML4~Tableには各階層のエントリのインデックス(9bit)

- 各テーブルの指定されたインデックスをたどる

- 最終的な物理アドレスをPage Tableから得る

x86-64 LinuxのMMUの仕組み



- 仮想アドレスが，0x7ffffffe358の場合
- PML4 → 255
- Pointer → 511
- Directory → 511
- Table → 510
- Offset → 856

- Tableの39-12bitの値(4KB境界)にOffsetの値を足す
- **→物理アドレス算出**

実装

- 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
- CR3の値を**監視対象ホストから通知**
- 通知された値を起点に，4階層の値を取得
  - 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

- PML4, Pointer, Directory, Tableの各エントリから，次の階層のアドレスを取得

- Page Tableまで到達したら，物理アドレスの保持(Page Tableの値まではポインタ)

- 物理アドレスを直接指定し，**実際の値**を保持

実装

- 以下の構造体にデータを格納

```
typedef struct _BINARY
{
    uint64_t pml4Index;
    uint64_t pdpIndex;
    uint64_t pdIndex;
    uint64_t ptIndex;
    uint64_t offset;
    uint64_t addr;
    uint64_t virtAddr;
    uint64_t value;
} BINARY;
```

実装

- 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

- 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監視対象とする

評価

- 監視するプロセス
- https://github.com/dooooooooingggggg/getcr3/blob/master/user.c
  - 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
  - CR3の値をカーネルモジュール経由でprintf()
- 現在のUNIX TIMEをprintf()

評価

- 監視対象ホスト

```
$ ./user
# .
# .
# .
# .
# a:1548316892(0x5c4970dc) at 0x7fff6509aa90
# 255 509 296 154 2704
# CR0: 0x80050033
# CR2: 0x7fbf309c3230
# CR3: 0x274be2000
# CR4: 0x160670
```

評価

- 監視対象ホスト・別コンソール

```
$ sudo su -
root$ echo c > /proc/sysrq-trigger
```

- 監視対象ホストでカーネルパニックを起こす

評価

- リモートホスト(本研究の実装)からメモリの読み込みを開始

- https://github.com/doooooooooingggggg/dmaLinux/blob/master/src/dmaLinux.c

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]) : 0x80050033000007fff
# 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(次スライドに監視プロセスを再掲)

- **5c4970de**(= **1548316894** (UNIX TIME))にカーネルパニックが発生

評価

- 監視対象ホスト(再掲)

```
$ ./user
# .
# .
# .
# .
# a:1548316892(0x5c4970dc) at 0x7fff6509aa90
# 255 509 296 154 2704
# CR0: 0x80050033
# CR2: 0x7fbf309c3230
# CR3: 0x274be2000
# CR4: 0x160670
```

評価

- CR3の値が与えられるという限定条件下において,
  カーネルパニック時に仮想アドレス空間を復元

今後の展望(卒論に向けて)

- CR3レジスタはtask_struct->mm_struct->pgdに格納
  - カーネル空間にあるこの変数を外から探す
  - 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