



Livegrid™



# セキュアVMの構築

Intel と AMD の比較、あともうひとつ...

Tsukasa Ooi <[li@livegrid.org](mailto:li@livegrid.org)>  
Livegrid Incorporated, Lead Analyst



Livegrid™



## 関連する事項

- 仮想化
- ふるまいベースのマルウェア検出
- リバースエンジニアリング



## セキュアVMとは？

- “セキュリティ拡張されたVM” のほうが正しいかも...
- セキュリティ拡張された機能
  - 不可視のブレークポイント
  - 直接のメモリ参照/改変
  - システム I/O の監視
  - 命令/メモリトレース
- これらはセキュリティ目的に有用である!
  - メモリ参照による rootkit の検出
  - システム I/O の監視によるマルウェアの検出



## どう仮想マシンを“保護”するか？

- (CPU の仮想化支援機能がある場合)  
おおまかに言えば 2 つの方法がある。
- デバッグレジスタを使う (DR0~DR7)
  - DR7.GD bit : DRx レジスタにアクセスできなくなる
  - DRx アクセス時の VM exit : 読み書きを監視できる
- ページテーブル改変を使う
  - PTE.R/W bit : ページ書き込みを監視できる
  - PTE.P bit : ページへの全てのアクセスを監視できる
  - PTE.NX bit : ページの実行アクセスを監視できる



## ブレークポイントの設定 (Debug Registers)

- VM exit を次のアクセス時に設定する:
  - DRx へのアクセス
  - #DB (デバッグ) 例外の発生
- ゲストの DR7.GD bit をオンにする
- ゲストのデバッグレジスタを用いてBPを設定
  - カーネル関数やシステムコールに設定するのが有用
- ブレークポイントがヒットした場合...
  - 追加の処理を行う



## ブレイクポイントの設定 (Page Table)

- VM exit を次のアクセス時に設定する:
  - #PF (ページフォルト) 例外
- ページの PTE.P bit か PTE.NX bit を 0 にする
  - ゲストのページテーブルを変更せず、Shadowを変更する
- ページフォルトが起こった場合:
  - ブレイクポイントがヒットしたかチェックする (false-positive が考えられるため)
  - ブレイクポイントがヒットしているなら、追加の処理を行う



## 比較 (Debug Registers / Page Table)

- デバッグレジスタの使用はシンプルである
  - 実装も簡単。
- ページテーブル改変は5つ以上のBPをセットできる
  - デバッグレジスタを使用するだけでは 4 つに制限される
  - もしゲストがデバッグレジスタを使用するなら、その動作をページテーブル改変でエミュレートしなければならない
- デバッグレジスタの使用は高速である
  - オーバーヘッド低, 別箇所における false-positive 無し
- ページテーブル改変は柔軟である
  - 使い方次第で何でも監視できる



## CPU による仮想化支援機能の考察

- いくつかの重要な命令をフックできない
  - sysenter, sysexit など
- バイナリ変換と比べると制限された監視手法



## AMD-V/Intel VT-x の比較

- AMD-V だけが IRET 命令をフックできる
  - ゲスト OS は EFLAGS.RF フラグをオンにして IRET 命令を実行することで、ホストの BP を回避できる (EFLAGS.RF : 次の 1 命令だけ BP のヒットを抑制する)
- デバッグレジスタを使うなら、AMD が better
  - ページテーブル改変を使う場合、何の機能差もない
  - Intel でも、システムコールを監視するような場合は十分 (IRET 命令で呼ばれることがほぼ無い)



## Second Level Address Translation (SLAT)

- 仮想化における新しい機能
- アドレス変換において 2 つのページテーブルを使用
  - Guest Logical Addr → Guest Physical Addr
  - Guest Physical Addr → Host Physical Addr
- Intel と AMD の双方が SLAT 拡張を実装している
  - Intel : Extended Page Tables (EPT)
  - AMD : Nested Paging / Rapid virtualization index (RVI)
- しかしこれらには問題がある...



## SLAT に関する考察

- さらに限定される監視手法
  - ゲスト OS の “本当の” ページテーブルが可視である
  - 2 つのページテーブルの参照は、(一部の) セキュリティ目的においては悪いことである
  - 最悪の場合、SLAT 機能を有効にできない!
- デバッグレジスタはまだ使用できる



## SLAT の機能差については？

- Intel EPT においてはページを“実行専用”にできる
  - CPU によってサポートされていれば...
    - 今のところ、Intel EPT をサポートしている全ての CPU でこの“実行専用”ページが使用可能
  - ステルス性は十分ではないが、ほぼ不可視のコード領域を作り、そこを実行できる
  - ただしゲスト OS もその部分を jump 系命令で実行可能であることに気をつけよ
- ほとんど同じだが、敢えて挙げるなら Intel



Livegrid™



THANK YOU?



Livegrid™



あともう1つ

# X86\_64 アーキテクチャにおける X86 CPU の完全仮想化



Livegrid™



## x86\_64 での x86 エミュレーション

- もちろんバイナリ変換を使う...
- これらは可能な上に、ほとんど実用的!



## なぜ “x86 on x86\_64”?

- アーキテクチャがとても似ている
  - 通常の x86 命令なら、x86\_64 でも最大 2 命令でエミュレートすることができる
- 追加メモリが存在する
  - x86 をエミュレートするには 4GB のアドレス空間が必要だが、x86\_64 においては全く問題のない量である
- 追加レジスタがある!
  - 追加された 8 つのレジスタを、コンテキストなどを保存するために使用することができる!



## どう x86 を “emulate” する? (1)

```
PUSH EBP
```

```
MOV EBP, ESP
```

```
MOV EAX, [EBP+8]
```

```
MOV EDX, [EBP+12]
```

```
MOV EAX, [EAX*4+EDX]
```

```
POP EBP
```

```
RET
```



## どう x86 を “emulate” する? (1)

PUSH EBP

MOV EBP, ESP

MOV EAX, [EBP+8]

MOV EDX, [EBP+12]

MOV EAX, [EAX\*4+EDX]

POP EBP

RET

MOV [RCX+RBP-4], EBX

LEA EBP, [EBP-4]

MOV EBX, EBP

MOV EAX, [RCX+RBX+8]

MOV EDX, [RCX+RBX+12]

LEA R14D, [EAX\*4+EDX]

MOV EAX, [RCX+R14]

MOV EBX, [RCX+RBP]

LEA EBP, [EBP+4]

(Return Intrinsics)



## どう x86 を “emulate” する? (2)

- レジスタの再割り当て
  - ESP → EBP/RBP  
(x86\_64 ではスタックが 8 バイト単位である)
  - EBP → EBX/RBX (スタック参照のロスを減らす)
- RCX : 32 ビット仮想アドレス空間のベース
  - 全ての仮想アドレス!
- R14/R14D : 一時レジスタ (例ではアドレス計算に使用)
- R15 : エミュレーションシステムで予約



## バイナリ変換を使って...

- x86 のメモリ/命令完全トレース
  - 全ての実行パス
  - メモリ read/write
  - I/O 処理
- オーバーヘッドは高いが、やる価値はある



## 何のために？

- “干渉”を検出する
  - 脆弱性の利用
  - フック
  - (テーブルやメモリ構造体の) 改変
- リバースエンジニアリング
  - “全て”をトレースできる!
    - アンチリバースエンジニアリングさえも検出可能
  - プロトコル解析 (ネットワーク, ファイル...)
  - アルゴリズム解析および検索
    - アルゴリズムの検索は Anti-DRM に有用かも



# どう x86 をトレースつきで “emulate” する? (1)

PUSH	EBP	MOV	[RCX+RBP-4], EBX
		LEA	EBP, [EBP-4]
MOV	EBP, ESP	MOV	EBX, EBP
MOV	EAX, [EBP+8]	MOV	EAX, [RCX+RBX+8]
		MOV	[R13], EAX
MOV	EDX, [EBP+12]	LEA	R13, [R13+4]
		MOV	EDX, [RCX+RBX+12]
MOV	EAX, [EAX*4+EDX]	MOV	[R13], EDX
		LEA	R13, [R13+4]
POP	EBP	LEA	R14D, [EAX*4+EDX]
		MOV	EAX, [RCX+R14]
RET		MOV	[R13], EAX
		LEA	R13, [R13+4]
		MOV	EBX, [RCX+RBP]
		MOV	[R13], EBX
		LEA	R13, [R13+4]
		LEA	EBP, [EBP+4]

(Return Intrinsics)



## どう x86 をトレースつきで “emulate” する? (2)

- R13 : トレースを出力するアドレス
  - メモリ読み取りの結果
  - 分岐先の命令 (EIP)
  - これだけをトレースすることで、全てが判明する!
- 通常の x86 命令を実行するために、  
x86\_64 においては 1~5 命令を実行する
  - バイナリ変換プログラムが最適化されれば、  
もっと効率が上がる!



## 実用的なのか？

- そこそこ。
  - ディスク書き込みのストール抜きでのパフォーマンス指標:  
Pentium 4 1.5GHz  
≥ **エミュレーション中の Core i7 3.0GHz**  
> Pentium III 1.0GHz
- システムエミュレーションには十分ではないが、ユーザモードエミュレーションには十分である!
- 他のどのトレース手法よりも高速  
(トレース機能付きのデバッガなど...)



## 制約 / 考察

- x86 セグメンテーションとアドレッシング
  - x86\_64 は “フラットメモリモデル” に制約されている
  - セグメントのリミットを模倣するには多大な計算量が必要
- 超高速なディスクが必要
  - 幸いにも書き込みは “シーケンシャル” なので、HDD を使った RAID-0 が有用かも
- 大量のリソースを必要とする
  - 特に主記憶容量について顕著



## いつ使用可能になる？

- 今のところ、実験的なプログラムが動作するのみ。
  - 実用的プログラムをエミュレートするには  
至っていない
- 2010 年前半までには公開できれば...



Livegrid™



何か質問は？

**THANK YOU.**