SecHack365

Rune

～CPUの特権機能をユーザープログラムに安全に公開する仕組み～

学習駆動コース　坂井ゼミ　三浦大輝

概要

RISC-VというCPUアーキテクチャ上のLinux環境を対象として、CPUの特権命令をユーザープログラムが安全に使えるようにする**仕組み(Rune)** と、Runeを使った**アプリケーション**としてサンドボックスを作成した。

Rune

特徴

・インストールが簡単
　・カーネルモジュールをインストールするだけ。
　　既存のカーネルに手を加える必要がない。
・用途が幅広い
　・Runeを使ってアプリケーションを作ることができる。
　　・Rune自体は仕組みを提供するだけ。Runeを使って何をやるかはユーザー次第。
・RISC-Vに対応している
　・今後は他のCPUアーキテクチャも対応予定だが、最初にRISC-Vに対応した。
　・RISC-Vのハイパーバイザー拡張を利用して何か便利な仕組みを作った事例は、調べた限りでは存在しない

使い方

1. Linuxカーネルを起動
2. Rune (カーネルモジュール) をダウンロード
3. $ insmod rune
4. ユーザープログラムからRuneを使用

```c
#include "librune/rune.h" // (1)
int main(int argc, char *argv[])
{
    rune_init_and_enter();  // (2)
    asm volatile("csrr a0, sepc\n");  // (3)
    return 0;
}
```

(1) Runeのライブラリをinclude
(2) CPUの特権命令を使う前にrune_init_and_enter()を呼び出す
(3) 特権命令を呼び出す

Runeに脆弱性があったら？

Runeの構成要素は以下の2つ。

・libRune (ユーザーがRuneを利用するためのライブラリ)
　・dev/rune にioctlを送って、U-modeとVS-modeの遷移を担う関数 (ioctlのラッパー関数)
・カーネルモジュール
　・/dev/runeにioctlが送られたら、引数に応じてモード遷移を行う。

libRuneに脆弱性があったらどうなる？

=> 問題なし。libRuneはVS-modeやVU-modeでのみ動くプログラムなので、もし脆弱性があってもVS-modeの外に抜け出すことができないためHS-modeのコントロールレジスタ、ページテーブルなどを触ることはできない。一般的なハイパーバイザーで、ハイパーバイザーの実装に脆弱性がなければゲストOSやゲストOS上のアプリケーションに脆弱性があってもホストOSには影響が及ばないのと同じ理屈。

カーネルモジュールに脆弱性があったらどうなる？

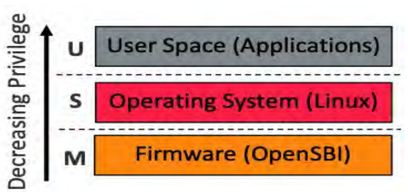
=> 問題あり。一般的なハイパーバイザーでハイパーバイザーの実装に脆弱性がある状態や、ハイパーバイザーが関係ない (普通の) 環境で、OSの実装に脆弱性がある状態と同じ危険度。カーネルモジュールにおける脆弱性対策に関して、調べた限りでは一般的に有効な解決方法はなさそうだった。しかし、以下のような面白い研究があった。

A Policy-Centric Approach to Protecting OS Kernel from Vulnerable LKMs , https://onlinelibrary.wiley.com/doi/am-pdf/10.1002/spe.2576

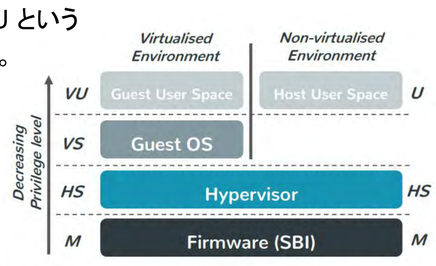
Fast Byte-Granularity Software Fault Isolation, https://www.sigops.org/s/conferences/sosp/2009/papers/castro-sosp09.pdf

仕組み

RISC-V の「ハイパーバイザー拡張」という拡張機能を利用。
この拡張機能を有効にするとVS と VU という
2つの新たな特権レベルが追加される。



ハイパーバイザー拡張が無効な時の特権レベル

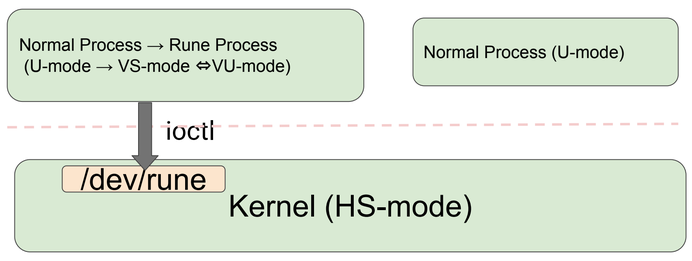


ハイパーバイザー拡張が有効な時の特権レベル

ハイパーバイザー拡張を有効にすると新たな特権モード (VS, VU) を利用できる。このモードは、一般的にはRISC-V上でハイパーバイザーを実装する際に用いられる。

VSモードはハイパーバイザー上のゲストOSを動かす特権モードで、VUはゲストOS上のユーザープログラムを動かす特権モードである。

ユーザープロセスをこの新たな特権モード (VS) に遷移させることで特権命令を安全に使えるようにする。(一般的なハイパーバイザー上のゲストOS が特権命令を使えるのと同じ理屈。)



カーネルモジュールを読み込む /dev/runeというデバイスが新たに作成され、このデバイスにユーザープロセスがioctlを送ることで、VS-modeに遷移し特権命令が使えるようになる。

Runeを使ったサンドボックス

特徴

・ゲストOSの起動が不要なので高速に起動できる
・システムコールの制限
　・サンドボックスのユーザーは以下のようなコードを記述することでシステムコールの取り扱い方を設定できる

```c
static int syscall_monitor(struct rune_tf *tf)
{
    switch (tf->rax) {
    case SYS_execve:
        printf("[sandbox]: execve() was called!\n");
        exit(1);
    }
    return 1;
}
```

　・上のように設定したサンドボックス上でexecve() の呼び出しを検知・制限している様子

```
~/src/rune/apps/sandbox $ sudo ./sandbox ./evil_binary
[sandbox]: execve() was called!
```

・システムコールの処理の高速な差し替え
・シンプルな実装 (1500行)

他のサンドボックスとの比較

・ホストOSに仮想化ソフト (Virtual Box など) をインストールする場合
　・🤢 ゲストOSの立ち上げが必要なので起動が遅い。
　・🤢 システムコールを差し替えようとすると、仮想化ソフトのソースコードをいじらないといけない。

・ハイパーバイザー (kvm, Hyper-V など) を使った場合
　・🤢 ゲストOSの立ち上げが必要なので起動が遅い。
　・🤢 システムコールを差し替えようとすると、ゲストOSかハイパーバイザーのソースコードをいじらないといけない。

・コンテナ (Docker) を使った場合
　・😃 ゲストOSの立ち上げが不要なので起動が早い。
　・🤢 seccompでシステムコールの制限はできるけど挙動の差し替えはできない。

・コンテナ (Docker) + ptraceを使った場合
　・😃 ゲストOSの立ち上げが不要なので起動が早い。
　・🙂 システムコールの差し替えを実現はできるが遅い。

・Runeを使った場合
　・😃 ゲストOSの立ち上げが不要なので起動が早い。
　・😃 システムコールの差し替えを高速に実現できる。

システムコール差し替えのベンチマーク

ptrace と Runeを使ったサンドボックス のシステムコールの処理の差し替えパフォーマンスを比較した。

【測定対象】
“foo.txt” というファイルを open() する処理を“bar.txt” というファイルの open() に差し替える際に、ファイルを open() してから close() するまでの時間を測定。
システムコールの差し替えなし、ptraceによるフックを用いた差し替え、Runeサンドボックスによる差し替え の3パターンを比較。

【測定結果】

差し替え手法	ファイルをopen()してからclose()するまでの時間 (10回の平均)
差し替えなし	43.6 [μs]
ptraceによるフック	80.1 [μs]
Runeサンドボックスによるフック	52.0 [μs]

ptraceよりも高速なシステムコールの差し替えを実現した！

・Rune と Runeを使ったアプリケーション を実装した。
・Runeを使ってサンドボックスを実装すると、既存のものより 起動 とシステムコールの差し替え が高速なものになった。
　・他のアプリケーションを実装してみたらまだ見ぬメリットがあるかもしれない。
・次はRuneを使ったコンテナランタイムを実装したい。

NICT 国立研究開発法人 情報通信研究機構