SecHack365

COVERAGE-GUIDED FUZZING FRAMEWORK FOR RISC-V SBI FIRMWARE

Hisame

RISC-V 最高特権ファームウェアに対するファジングフレームワーク

西坂 龍太郎
学習駆動コース 坂井ゼミ
SecHack365 - 2025

SecHack365での歩み

ハイパーバイザ開発から、SBIファジングへ

2月 – 4月 自作OS開発
MikanOS / WasabiOS 等を利用し
低レイヤ開発の基礎を学習

5月 – 11月 Tsuyu 開発
仮想化支援機構を利用する
一般的な RISC-V Type-1 Hypervisor

12月 – 1月前半 Akisame 開発
仮想化支援機構の無い環境でも動かせる
新たな設計の M-mode Hypervisor

1月後半 – 2月 ★ Hisame 開発
M-mode SBIファームウェア に特化した
ファジングフレームワーク

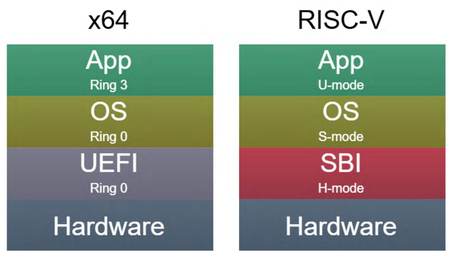
RISC-V実機の大半は 仮想化支援機構 **未搭載**
→ なしでもハイパーバイザを動かせないか？

M-modeの安全性が **全体のセキュリティの前提**
→ その安全性をどう保障するか？

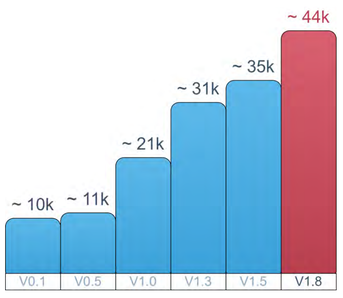
背景と課題

SBIファームウェアのリスク

SBIファームウェア
- ハードウェアとOSとを仲介
- RISC-Vの**最高特権（M-mode）**で実行
- 脆弱性があれば、**システム全体へ影響**



OpenSBI: SBIのリファレンス実装



バッファオーバーフロー
tbufのタイミング不備による
範囲外書き込み

Nullポインタ参照
デバッグトリガ拡張における
Nullポインタ参照

Null終端の欠如
特定のバッファサイズでの
範囲外読み込み

入力値チェック欠落
FWFT拡張における
入力チェックの欠落

コード規模 - **6年で4倍以上に**

既に **複数の脆弱性** が発見されている

課題

現状	有効な手法	ギャップ
脆弱性探索は **人手ビュー** に依存	**ファジング** は Appで広く成功	**SBI向けファザー** は 存在しない

提案

Hisame – SBI Fuzzer

RISC-V SBIファームウェアに対し、Coverage-Guided Fuzzingを適用する
これまでSBI Fuzzingを妨げてきた3つの障壁を解決し、効率的に脆弱性探索

SBI Fuzzing - 3つの課題と解決

課題	Hisameの解決策
入力の定義 何を入力として ファジングするか	**ecall構造化入力** SBI呼び出し（**ecall**）を入力として利用。専用ハーネスがecallを実行する SBIで頻出の定数等を重点的に利用する
計測と分離 M-modeの状態を どのように計測するか	**LibAFL QEMU + KASan + QASan** **ブロックフック** でM-modeのプログラムのカバレッジを計測 KASan / QASan で、SBIファームウェアを書き換えず、脆弱性を検知
速度 ファジングを左右する 高速化をどう達成するか	**スナップショット + 並列化** 起動後のSBIファームウェアの状態をスナップショット、重い初期化処理を **スキップ** 複数コアを利用した、並列ファジング

Hisameの利点

01	02	03
脆弱性探索の自動化	**効率的な経路探索**	**導入の容易さ**
CI統合で24/365テスト 人手で見落しやすいバグも発見	カバレッジに基づいた 未到達領域の重点的探索	実機不要 SBIのコードの変更不要

実装

Hisame アーキテクチャ

Hisame-driver (host)

LibAFL QEMU インプロセス実行	**ブロックフック** M-mode カバレッジ	**CmpLog** 比較値学習
QASan 動的メモリ検査	**KASan** 静的計装検査	**Triage** 例外分類

QEMU (riscv64)

OpenSBI（KASan計装対応） M-mode（ターゲット）	**Hisame-harness** S-mode（入力適応）

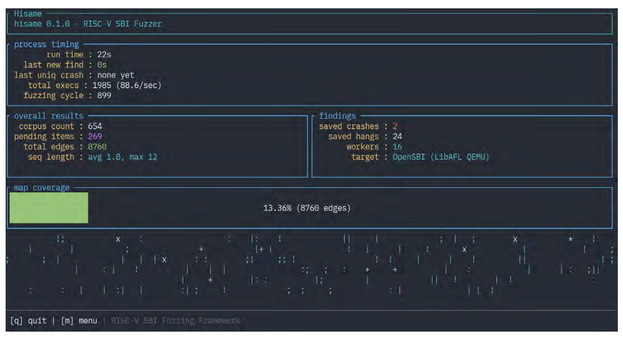
主要な技術的工夫

⛶ 2種類のサニタイザ
KASan（コンパイル時計装）と **QASan**（QEMU動的フック）を併用
スタック / ヒープのオーバーフロー等を検知

⇌ Pointer-aware Mutator
ecallには、引数としてポインタを要求するものが多数存在
有効なアドレス を自動生成し、引数として利用する

⊕ CmpLog
比較命令のオペランド を実行時に記録
SBI内部で利用されるマジックナンバーや定数チェックを自動で突破

デモ画面



意図的に脆弱性を仕込んだOpenSBIにおいて、
クラッシュを発見することに成功

まとめ

課題

- SBIファームウェアは **最高特権** で動作
- 近年、複数の脆弱性が発見されている
- ひとたび攻撃を受ければ、システム全体が危殆に
- 現状の脆弱性探索は人手に依存し、**非効率的**

提案

- **Hisame** を提案 / 実装
- ファジングによって、**自動で**脆弱性を探索
- KASan / QASan によって、メモリバグを検知
- CmpLog / Pointer-aware Mutatorで効率化
- スナップショット / 並列化で高速化

展望

- 大規模ファジングで実際の脆弱性を発見する
- RustSBI等の他SBI実装への対応
- RISC-Vエコシステムの **安全性向上** に貢献

NICT National Institute of Information and Communications Technology