# プロセッサトレースを用いた組み込みデバイス向けファザーの開発 ー 高速な ARM 向けファザー ー

## 1　背景

脆弱性の発見は，これまで卓越したハッカーの天才的能力によるところが大きかったが，近年では入力空間を網羅的に探索して脆弱性を発見するファザー（Fuzzer）が普及しつつある．脆弱性が問題となるサーバや個人用 PC の多くはインテル製プロセッサが使用されており，従来のファザー研究もインテル製プロセッサを前提とするものが大半を占めている．しかし，近年，スマートフォン，IoT や組込みデバイス機器の普及に伴い ARM 社製プロセッサをベースとするシステムの脆弱性検査のニーズが高まっており，Apple や Google がバグバウンティプログラムとして Google は 150 万ドル，Apple は 100 万ドルを脆弱性の対価として支払うと発表している．

## 2　目的

本プロジェクトでは組み込みデバイス機器，具体的には Android や iOS などの OS の脆弱性を自動で発見するためのファザーを開発することを目的とした．本プロジェクトの特徴は，これまでのファザー研究であまり注目されてこなかった ARM 社製プロセッサを対象とし，近年プロセッサに組み込まれはじめたハードウェアトレース機能（CoreSight）を用いることにより，ソフトウェアトレース機能を用いる従来手法と比較して大幅に脆弱性発見効率を高める点にある．

## 3　開発の内容

脆弱性は様々な要因により発生するバグを外部からの入力により誘発して，システムの正常な機能を麻痺させることができるというプログラムの性質である．図 1 左のプログラムには，システムがヒープ領域にあるメモリ A を解放した後，新たに獲得したメモリ B がメモリ A の領域を再利用した際に，過去のメモリ A のデータがメモリ B から参照できることから生じる脆弱性が含まれている．プログラムを見て脆弱性を発見することは一部の天才を除いてもはや不可能である．ファジングツー

```
size_t numProperties = propertyNames.size();
Vector<PropertyDescriptor> descriptors;
MarkedArgumentBuffer markBuffer;
for (size_t i = 0; i < numProperties; i++) {
    JSValue prop = properties->get(exec, propertyNames[i]);
    if (exec->hadException())
        return jsNull();
    PropertyDescriptor descriptor;
    if (!toPropertyDescriptor(exec, prop, descriptor))
        return jsNull();
    descriptors.append(descriptor);
```

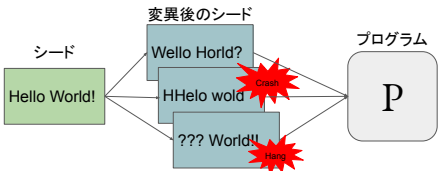


図 1:　（左）Use after free 脆弱性の例,（右）ファジングの動作原理

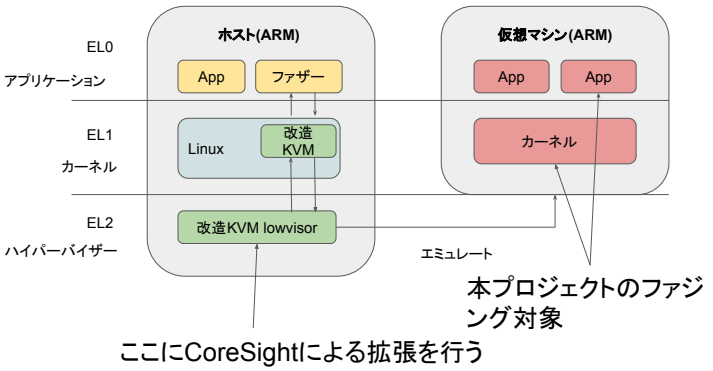


図 2: ZeroSight 概要

ルは図 1 右のように初期入力に交叉や変異を繰り返して脆弱性（バグ）を探索するツールである．

本プロジェクトでは，(1) 最新の Fuzzing 手法であるカバレッジベースの手法を用い，(2) ARM 社製プロセッサを対象として，(3) 一般に脆弱性の検査（ファジング）が難しいカーネルを対照するファジングツール “ZeroSight” を開発した．

本プロジェクトでは図 2 に示すように，ファジング対象となる仮想マシン（ARM）をホスト（ARM）上で動作させるために，改造 KVM（Kernel Virtual Machine）と改造 KVM lowviser（図の緑部分）を開発した．KVM は，CPU のハードウェア支援機能を使い，仮想マシン（ARM）の実行を高速化する．これに ARM プロセッサのハードウェアトレース機能である CoreSight を組み合わせることで高速にゲストマシン内のカーネルやアプリケーションのファジングができる．

しかし，図 3 の ARM 搭載 CPU ボードを調査・実験した結果，CoreSight の実装はまだ不完全で，ハードウェアトレースに ARM プロセッサの CoreSight 機能を利用できないことが判明した．このため本プロジェクトでは，QEMU を改造することでソフトウェアトレースによるファジング機能も並行して実装し，CoreSight が利用可能なハードウェアが入手可能になり次第，そこと差し替えられる構造とした（図 4）．

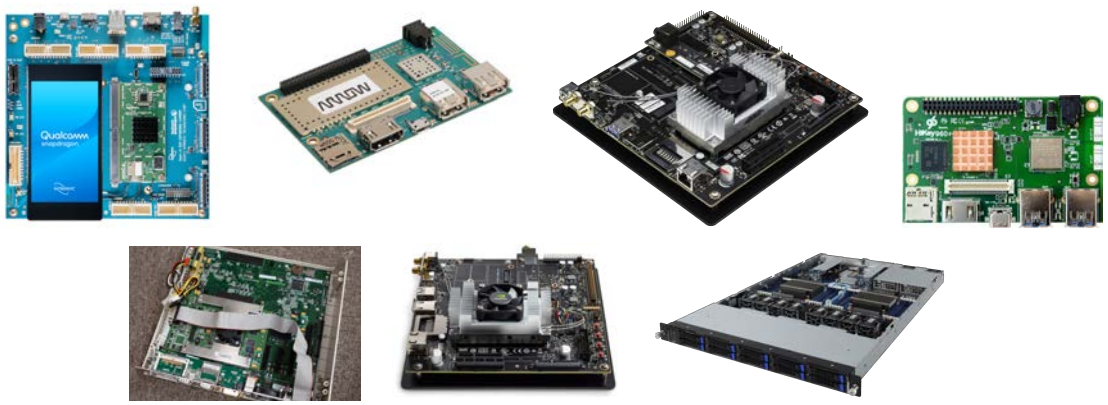

図 3: 本プロジェクトで CoreSight 機能を評価した ARM 搭載 CPU ボード

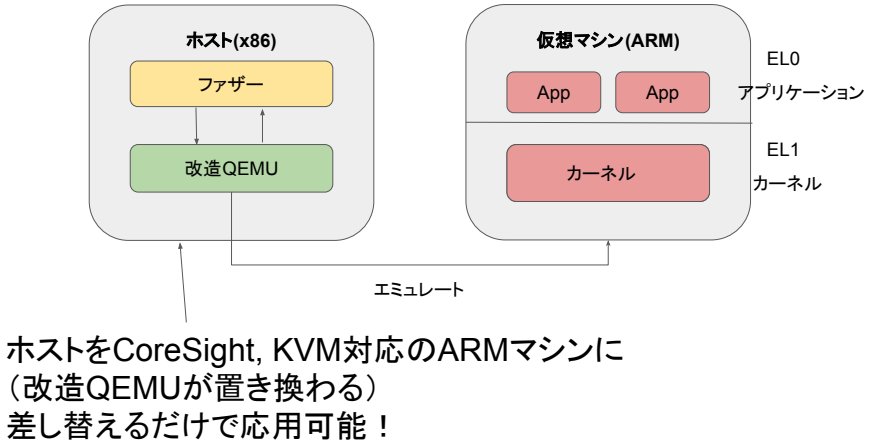


図 4: ZeroSight 概要（ソフトウェアエミュレーション版）

本構成に基づくカバレッジベースのファジングツールを ARM 版 Linux カーネルに適用し，326,881 個のバグを検出した．この数には，より深刻な 10,781 個のクラッシュ（カーネルパニック等で確認できる明確なバグ）と，316,200 個のハング（デッドロック等の応答遅延の結果，タイムアウトで判定されたバグ）を含む．10,781 個はクラッシュを誘発する入力の数に相当し，これは 69 個の異なる基本ブロックに含まれるバグが検出できたことを意味している．すなわち，ARM 版 Linux カーネルのバグを 69 個検出できた．

## 4 従来の技術との相違

本プロジェクトの特徴は，これまでのファザー研究であまり注目されてこなかった ARM 社製プロセッサを対象とし，ARM 社製プロセッサ上で動作するプログラムの脆弱性を検査するファザー “ZeroSight” を開発したことにある．

図 5 に示すように，ARM プロセッサは安価なことから多くのスマートフォンや IoT 機器に搭載されており，2016 年時点で既に CPU の出荷数は，Intel 製および AMD 製 x86 の 1000 倍以上を ARM 系プロセッサが占めている．脆弱性が問題となるサーバや個人用 PC の多くはインテル製プロセッサが使用されており，従来のファザー研究もインテル製プロセッサを前提とするものが大半を占めている.

ARM 系プロセッサにおけるファジングの課題は，安価であることの代償として処理性能が低いことから，ファジングの探索性能が x86 系プロセッサと比べて圧倒的に弱いことである．この問題は，ZeroSight の開発により，(1) CPU 組込のハードウェア支援トレース機能（CoreSight）を用いて，ファジング効率を圧倒的に向上させることができる, (2) QEMU による高速な x86 系 CPU で ARM システムのファジングを効率的に実行することができる，という 2 つの方法で解決した．


図 5: x86, ARM 出荷数（2007 年 ∼2016 年）
引用：Reddi, Vijay Janapa and Yoon, Hongil and Knies, Allan. “Two Billion Devices and Counting”. In: IEEE Micro 38.1 (Jan. 2018), pp.6-21

## 5 期待される効果

本プロジェクトの開発成果を活用すれば，スマートフォンやタブレットのみならずセキュリティカメラや家電製品などの IoT 機器に対してもファジングが可能になる．従来のファジングは，PC・サーバ等の x86 系プロセッサに限定されていた．今回の開発成果は ARM 系 CPU が搭載されたスマートフォンや IoT 機器等の圧倒的多数のデバイスのセキュリティ向上に貢献できる．

## 6 普及の見通し

本開発成果は，未踏 OB クリエータの木村廉氏が起業した株式会社リチェルカセキュリティが提供する脆弱性検査ツール「Wyvern Pro™」に採用される予定である．

## 7 クリエータ名（所属）

大塚 馨（早稲田大学高等学院）

## （参考）関連 URL

- 検出バグ情報：`https://github.com/roppinhoppin/kernel_crashes`