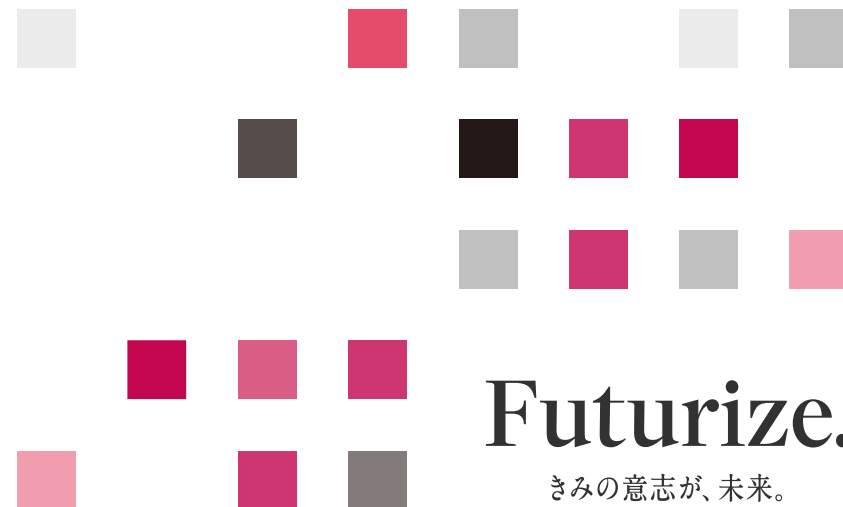


# MileSan: Detecting Exploitable Microarchitectural Leakage via Differential Hardware-Software Taint Tracking

著者： Tobias Kovats, Flavien Solt, Katharina Ceesay-Seitz, Kaveh Razavi

CCS 2025, Taipei, Taiwan





# 背景：CPUの脆弱性

- 投機的実行

- CPUの性能向上のため、次に実行される命令を予測し先回りして処理を開始

- Spectre：投機的実行を悪用し、情報を盗み出す


1. if文の条件が真となるような正常なx(array1の範囲内)を何度も与えて、「このif文は、だいたい真になるものだ」と学習させる
2. array1の範囲を超えるxを入力。CPUは「今回も真だろう」と誤って予測し、ifブロック内のコードを投機的に実行
3. その結果、不正なxを使って秘密情報（array1[x]）を読み込み、その値に応じたアドレス（array2[...]）にアクセスし、キャッシュに痕跡を残す。

- Meltdown<sub>(P15で補足)</sub>

- 権限の壁を越えて、カーネルの秘密情報にアクセスし痕跡を残す

いずれの攻撃も、キャッシュに残った痕跡に対し**メモリアクセス時間を計測**することで（Flush+Reloadなど）、秘密情報を特定

```
if (x < array1_size)
    y = array2[array1[x] * 256];
```



# 課題：CPUの脆弱性発見の難しさ

---

- 脆弱性発見にはファジングという自動テスト手法を使う
  - CPUの設計段階で、ファザーと呼ばれるツールがテストプログラムを大量に自動生成・実行し、脆弱性を網羅的に探す
- 既存のファザーには過学習という問題があった
  - ハードウェアへの過学習
    - 情報が漏えいしそうな特定の部品（キャッシュなど）を手動でタグ付けし監視していたため、想定外の部品からの未知の漏えいを見逃す
  - ソフトウェアへの過学習
    - Spectreなど既知の攻撃のテンプレートに基づくテストプログラムの生成により、パターンから外れた未知の種類の脆弱性を見つけにくい
- 原因：なぜ過学習が起きてしまうのか
  - CPU内部の膨大な情報フローの中から、本当に危険で悪用可能なものだけを識別する汎用的なメカニズムがなかったため



# 提案手法

- 基盤技術MileSanを活用した、ファザーRandOSを提案
- MileSanの役割（基盤技術）
  - 検知器として：悪用可能な漏洩が起こす「実行時間の変化（PCへの影響）」のみを監視することで、ハードウェア過学習を解決
  - 生成支援として：TAISS技術により、テンプレートに依存しない安全なプログラム生成を可能にし、ソフトウェア過学習を解決
- RandOSの役割（全体システム）
  - MileSanを検知器および生成支援として活用し、権限レベルやメモリ空間を自由に横断する複雑なテストプログラム（ランダムOS）を生成・実行



# MileSanの基本概念

- テイント：追跡対象の機密情報
- AIFs：ISAをもとにした情報の流れ

add s0, a0, a1

beq a0, a1, target\_addr

図1

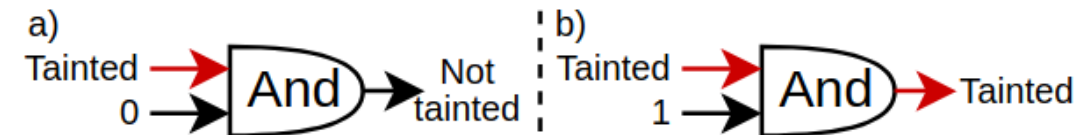
図2

- 明示的フロー：入力レジスタの値が出力に直接影響を与える(add命令)
- 暗黙的フロー：レジスタの値に基づいてPCが決定 (beq命令)

- MIFs：回路レベルでの実際の流れ

- キャッシュヒット/ミス、投機的実行

- (a)：入力の一方が0なため、もう一方がtaintedであっても出力は常に0



- (b)：入力の一方が1なため、もう一方の入力のtaintedにより出力もtainted ← (AIFsでは想定していないが、MIFsで特定のデータをきっかけに意図しない情報の流れが発生)




# MileSanの基本原理

---

- CPUの「設計図」と「実際の動き」を比較し、その差分から脆弱性を見つけ出す差分解析
- 差分解析のステップ
  1. ソフトウェアで設計図の流れ (AIFs) を追跡
  2. ハードウェアで実際の流れ(MIFs)を追跡
  3. 両者を比較し、悪用可能な差分 (漏洩) を検出←次スライドで解説





# MileSanの基本原理

MileSan foundation:  $MIO = \text{obs.} [ MIF - AIF ]$

- 悪用可能な差分（漏洩）を絞り込む
  - AIFsとMIFsの差分のすべてが危険なわけではない
  - 本当に悪用可能な差分は最終的に命令の実行時間に影響を与えるもの
- PCを監視して、悪用可能な情報漏えいを特定する
  - 具体的には、PCの値（アドレス）そのものではなく、ハードウェア（RTL）のテイント追跡におけるPCの値が確定するタイミングを監視
  - 「AIFsでは絶対にテイントされないはずのPCが、MIFsではテイントされた」←脆弱性発見
  - 膨大な情報の中から悪用可能な漏洩（MIOs）だけを効率的に検出可能

(P16で補足)

# MIOs発見の課題

- テイント爆発

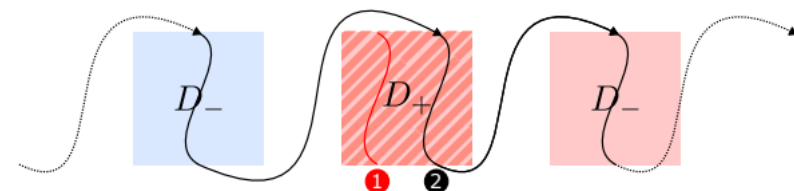
```
beq a0, a1, target_addr
```

1. CPUの設計図にはない、意図しない微細な漏洩（MIOs）が原因で、PCがテイントされる特定の瞬間（＝脆弱性）をピンポイントで見つけたい
2. レジスタa1にテイントが含まれていると、プログラムの実行経路(PC)が「意図された流れ（AIFs）」としてテイントされてしまう。
3. この広範囲なノイズ（意図されたテイント）の中に、本当に探したい信号（意図しない漏洩によるテイント）が埋もれてしまい、区別が不可能

- TAISS：テイント爆発を防ぐため、テストプログラムの生成プロセスを監視し、安全な命令だけを追加するように誘導(P15で補足)

- ① テイントされたデータの流れ
  - beqなどPCをテイントしうる命令を禁止。算術命令などは許可し脆弱性を誘発
- ② テイントされていないデータの流れ
  - どんな命令でも可

①の機密情報が②の実行経路の決定に影響を与えないように、追加できる命令を監視・制限







# RandOS : ランダムなOS

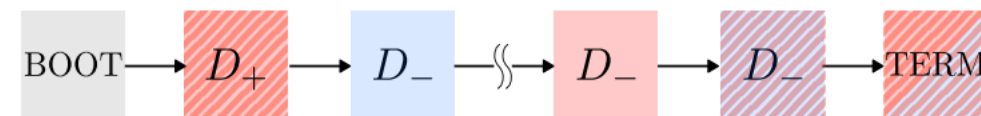
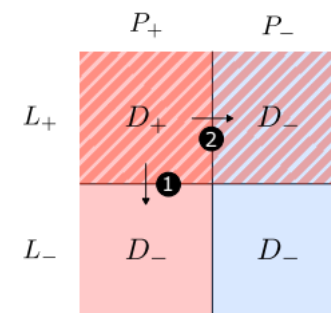
## • RandOS

- TAISS技術を使い構築した新しいファザー(ファジングをするツール)
- コンセプト
  - 現代の脆弱性は、権限レベルの境界など、複雑な状況下で発生
  - →そのような状況を再現できるOSのようなテストプログラムが必要

- P(Privilege)<sub>+</sub> : 機密データにアクセスできる高い権限 (カーネルモードなど)
- P<sub>-</sub> : アクセスできない低い権限 (ユーザーモードなど)
- L(Layout)<sub>+</sub> : 機密データが配置されているメモリ領域
- L<sub>-</sub> : 配置されていない領域

以下のような本来のルールを破って発生する不正な行動を何度も起こす

- ① メモリ領域の境界を超える漏えい(Specterなど)
- ② 実行権限の境界を超える漏えい(Meltdownなど)





# RandOSとMileSanの連携

---

## 1. プログラム生成

- TAISS技術を使い、テイント爆発を避けながら安全かつランダムなOSを生成

## 2. プログラム実行

- 生成したOSを、テイント追跡機能を持つCPUシミュレーター上で実行

## 3. 漏えい検出

- 実行中にMileSanがPCへの不正な情報の流れを監視し、脆弱性を検出

この連携により、従来手法の過学習の問題を解決し、未知の脆弱性の効率的な発見が可能に





# 評価：既知の脆弱性発見にかかる時間

- 概要：提案手法のRandOSが、既知の脆弱性を発見するまでの時間を既存の最先端手法であるSpecDoctorと比較
- 結果：平均で4.5倍高速に脆弱性を発見した
  - ランダムで多様なプログラムを生成したことで、脆弱性を引き起こす条件をより早く満たせた

Vuln.	Mean	Median	Std.Dev.	Speedup <sup>†</sup>
Spectre-V1	5h35m	4h12m	3h52m	4.8x
Spectre-V2	7h50m	7h56m	4h40m	3.9x
Spectre-RSB	11h28m	11h28m	0h17m	/
Meltdown	6h10m	5h33m	4h33m	5.6x
Trans. Meltdown	6h59m	7h13m	4h12m	3.8x
cp-Spectre-V2	7h41m	8h58m	4h32m	/
MDS*	6h44m	5h34m	4h33m	/
<b>Mean</b>	<b>7h30m</b>	<b>7h16m</b>	<b>3h49m</b>	<b>4.5x</b>





# 評価：未知の脆弱性の発見

- 概要：RandOSがどれだけ多くの、そして多様な未知の脆弱性を発見できたか
- 見方
  - DUT (Device Under Test)：CPUの名前
  - Covert Channel：悪用されたCPUの部品
  - P+/P-：権限レベル。（例）{S}/{U}はSepervisorとのドメインを超えた情報漏えい
  - Previously found by：空欄や"/"になっているものに今回新たに見つかった脆弱性
  - CVE：CVEが記載されているものは、公式に認められた新しい脆弱性
- 結果
  - 19件の未知の脆弱性が見つかった

Vulnerability	DUT	Covert Channel	{P+}	{P-}	Previously found by	CVE
Spectre-V1	BOOM	DCACHE	{S,U}	{S,U}	[14, 23, 38]	/
Spectre-V1-TLB	BOOM	TLB	{S,U}	{S,U}	[23]	/
Spectre-V2	BOOM	TLB	{S,U}	{S,U}	[14, 23, 38]	/
Spectre-V2-TLB	BOOM	TLB	{S,U}	{S,U}	/	requested
Spectre-V4	BOOM	DCACHE	{S,U}	{S,U}	[14, 23]	/
Spectre-V4-TLB	BOOM	TLB	{S,U}	{S,U}	/	requested
Spectre-RSB	BOOM	DCACHE	{S,U}	{S,U}	[14, 23]	/
Spectre-RSB-TLB	BOOM	TLB	{S,U}	{S,U}	/	requested
Meltdown	BOOM	TLB	{S}/{U}	{U}/{S}	[14, 17, 23]	/
Trans. Meltdown	BOOM	TLB	{S}/{U}	{U}/{S}	/	CVE-2025-29343
cp. Spectre V2	BOOM	TLB	{S}/{U}	{U}/{S}	/	requested
MDS*	BOOM	TLB	{S}/{U}	{U}/{S}	[14]‡	/
Spectre-SLS	CVA6	TLB	{S,U}	{S,U}	/	CVE-2025-29340
cp-Spectre-SLS	CVA6	TLB	{S}/{U}	{U}/{S}	/	CVE-2025-29340
MDS	CVA6	TLB	{S}/{U}	{U}/{S}	/	CVE-2025-46004
Trans. MDS	CVA6	TLB	{S}/{U}	{U}/{S}	/	CVE-2025-46004
div†	CVA6	DIV	{M,S,U}	{M,S,U}	[20]	/
divu†	CVA6	DIV	{M,S,U}	{M,S,U}	[7, 20]	/
divw†	CVA6	DIV	{M,S,U}	{M,S,U}	[20]	/
divuw†	CVA6	DIV	{M,S,U}	{M,S,U}	[20]	/
rem†	CVA6	DIV	{M,S,U}	{M,S,U}	[20]	/
remu†	CVA6	DIV	{M,S,U}	{M,S,U}	[20]	/
remw†	CVA6	DIV	{M,S,U}	{M,S,U}	[20]	/
remuw†	CVA6	DIV	{M,S,U}	{M,S,U}	[20]	/
Spectre-V1	OpenC910	DCACHE	{S,U}	{S,U}	/	requested
Spectre-V1-TLB	OpenC910	TLB	{S,U}	{S,U}	/	requested
div†	OpenC910	DIV	{M,S,U}	{M,S,U}	/	CVE-2025-46005
divu†	OpenC910	DIV	{M,S,U}	{M,S,U}	/	CVE-2025-46005
divw†	OpenC910	DIV	{M,S,U}	{M,S,U}	/	CVE-2025-46005
divuw†	OpenC910	DIV	{M,S,U}	{M,S,U}	/	CVE-2025-46005
rem†	OpenC910	DIV	{M,S,U}	{M,S,U}	/	CVE-2025-46005
remu†	OpenC910	DIV	{M,S,U}	{M,S,U}	/	CVE-2025-46005
remw†	OpenC910	DIV	{M,S,U}	{M,S,U}	/	CVE-2025-46005
remuw†	OpenC910	DIV	{M,S,U}	{M,S,U}	/	CVE-2025-46005



# まとめ

---

- 背景

- 既存の脆弱性検出手法は、既知の攻撃パターンに過剰適合しており、未知の脆弱性を見逃す課題があった

- 提案手法

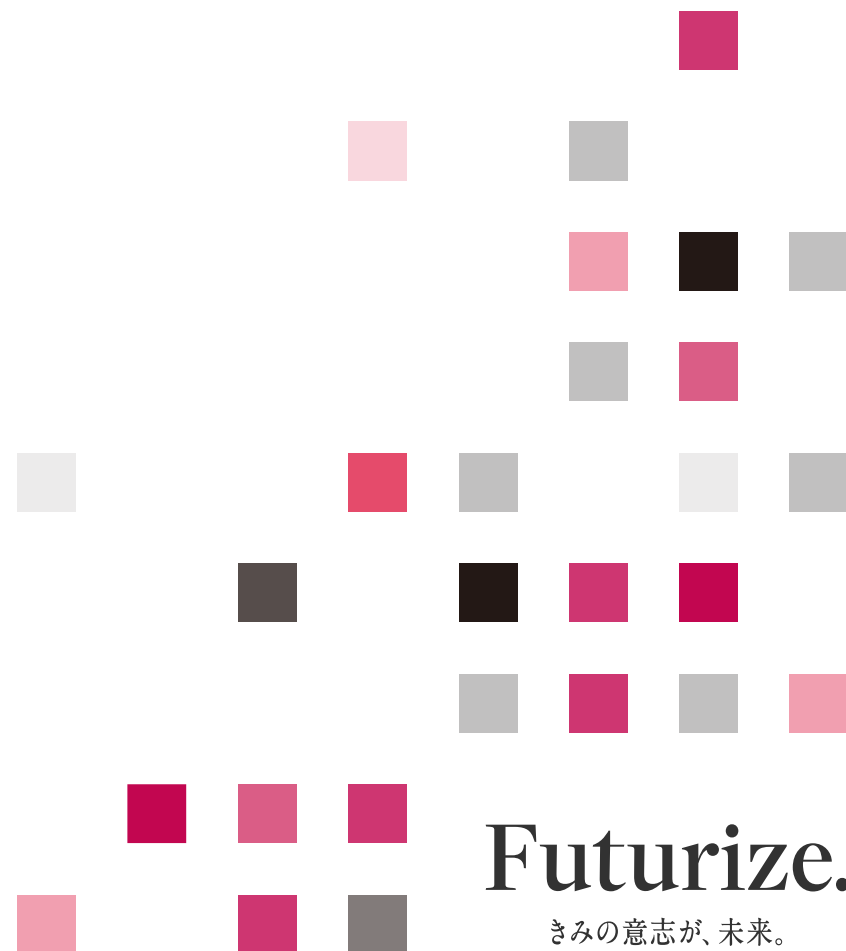
- CPUの設計図と実際の動きの差分から漏えいを検知するMileSanと、偏りのないテストOSを生成するファザーRandOSを提案した

- 結果

- 従来比で平均4.5倍高速に脆弱性を発見し、19件の未知の脆弱性を新たに発見することに成功した



# 補足資料





# Meltdown

- アウトオブオーダー実行を活用

- 依存関係（下図）

- 命令2は、命令1の結果（RAXの値）に依存している
    - 命令3は、他のどの命令にも依存していない
    - 命令4は、命令3の結果（RCXの値）に依存している

- 実行順序

- 長い時間がかかる命令1の前に命令3, 4を実行（命令2も後回し）

1. `MOV RAX, [address1]` ; メモリアドレス `address1` からRAXレジスタにデータをロードする
2. `ADD RBX, RAX` ; RBXレジスタにRAXレジスタの値を加算する
3. `INC RCX` ; RCXレジスタの値を1増やす
4. `ADD RDX, RCX` ; RDXレジスタにRCXレジスタの値を加算する

カーネルの秘密アドレス





# TAISSによるソフトウェア過学習の改善

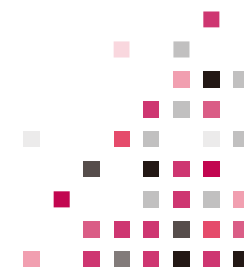
- 従来のファザー (SpecDoctor)

1. Spectre-V1のテンプレートを用意
2. 基本的な骨格を維持したまま、部分的な改変を加えて大量のテストプログラムを生成
  - 使用するレジスタの変更、array1のメモリオフセットをずらす


```
if (x < array1_size) {  
    // ... (optional other instructions)  
    y = array2[array1[x] * N];  
}
```

- TAISSによる変革

1. ゼロの状態からランダムに選択した命令を一つずつ追加
2. 1命令のたびにテイント爆発が起きないかチェック
3. 安全な命令のみを追加







# PC監視による脆弱性検出の詳細

---

- 前提

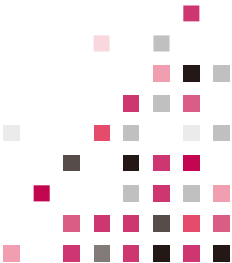
- div x0, x1, a0という除算命令（a0に機密情報（テイント））
- CPUが割る数（a0）の値によって計算にかかる時間が変わる特性を持つ

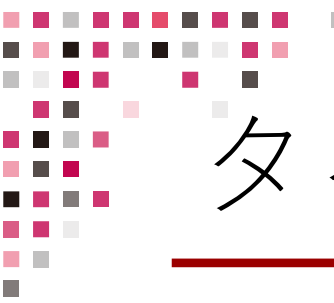
- AIFsでの分析

- div命令は、ISA（設計図）上では単なる算術演算
- ISAにはdiv命令は、入力の値によって実行時間が変わるという記述はないため、AIFsの分析では、この物理的な時間の変化を検知なし
- この命令はプログラムの実行経路を変更しないため、a0からPCへの情報の流れは存在しない

- MIFs

- a0の値（機密情報）によって、除算にかかる時間が変動
- 「a0の値」と「PCが更新されるタイミング」の間の依存関係を検知





# タイミングから検出できる脆弱性は？

---

- キャッシュタイミング攻撃だけでなく、あらゆるマイクロアーキテクチャ上のタイミングサイドチャネル攻撃を対象としている
- 実際に論文で発見されたのは、
  - TLB
  - 除算ユニット：命令の実行時間が入力される値によって変わる脆弱性
  - 分岐予測ユニット：分岐予測を意図的に操作して悪いコードを読ます