RowHammerを用いたRISC-V Keystone TEEに対する特権昇格手法の検討

西坂 龍太郎[1]　川崎 秀昌[2]　高前田 伸也[3,4,a)]　穐山 空道[2,b)]

概要：Trusted Execution Environment (TEE) は隔離された環境で安全にコードを実行する技術であり，医療や認証などセキュリティが重要なシステムでの活用が広がっている．TEE の安全性に関し，特定の TEE 実装では Spectre や Meltdown などの攻撃により保護された機密情報が搾取できることが知られている．一方で，TEE の安全性の RowHammer への耐性についてはほとんど議論が行われていない．本研究では，RISC-V の TEE である Keystone において，DRAM に保存されている管理情報が RowHammer によって攻撃を受けると特権昇格が発生すること，Context-Switch を大量に発生させることにより当該管理情報に対し RowHammer を引き起こせることを発見した．これらのアイディアをシミュレータ上で検証し，一定の条件下で実際に特権昇格できることを確認した．

Privilege Escalation Attack to RISC-V Keystone TEE via RowHammer

Ryutaro Nishizaka[1]　Hidemasa Kawasaki[2]　Shinya Takamaeda[3,4,a)]　Soramichi Akiyama[2,b)]

Abstract: Trusted Execution Environments (TEEs) provide isolated domains for secure code execution and are increasingly adopted in security-critical systems such as healthcare and authentication. It is known that some TEE implementations are vulnerable to Spectre and Meltdown, resulting in leakage of protected secret data. However, the resistance of TEE against RowHammer is hardly discussed as far as we know. In this paper, we point out that Keystone, a RISC-V TEE, stores some metadata in DRAM and an attacker can escalate their privilege by attacking it by RowHammer, and that triggering RowHammer to the metadata is plausible by issuing many Context-Switches. We evaluated these ideas in a simulation-based environment and confirmed that it is indeed possible to escalate the privilege under certain circumstances.

1. はじめに

近年，Trusted Execution Environment(TEE) の重要性が増している．TEE は，Enclave と呼ばれる隔離されて安全にコードを実行できる環境を提供する技術であり，Intel SGX，ARM TrustZone，RISC-V Keystone などの実装が存在する．Enclave は実装によるもののソフトウェア的な攻撃から物理的な攻撃まで幅広く耐性を持ち，OS やハイパーバイザなどを信頼せずとも安全に計算が可能だと考えられている．そのため，医療 [1]，認証 [2]，IoT 機器 [3]，機械学習 [4], [5] など幅広い分野で利用が進んでいる．

RowHammer は，DRAM の特定のアドレスへ繰り返しアクセスすることで隣接するデータが書き変わる攻撃 [6] であり，DDR5 などの新しいものを含めて多くの DRAM で実証されている [7]．RowHammer は対象機器への物理的なアクセスがなくとも実現可能であり，OS や VM などへの攻撃に利用されてきた．具体的には OS レベルの特権昇格 [8]，KVM における他の VM 内のデータの改ざん [9]，被攻撃者プロセスへのコード注入 [10] などの様々な応用が知られている．

このように RowHammer は非常に強力な攻撃であるが，これまで TEE の安全性の RowHammer への有効性は広く研究されてこなかった．我々の知る限り唯一の RowHam-

[1] 立教新座高等学校 Rikkyo Niiza High School
[2] 立命館大学 Ritsumeikan University
[3] 東京大学 The University of Tokyo
[4] 理化学研究所 革新知能統合研究センター RIKEN Center for Advanced Intelligence Project (AIP)
[a)] shinya@is.s.u-tokyo.ac.jp
[b)] s-akym@fc.ritsumei.ac.jp

mer による TEE への攻撃は Yeongjin らの研究 [11] である．この研究では Intel SGX 上で動作する Enclave に対して RowHammer を行うことで Denial-of-Service (DoS) 攻撃が可能であることを示した．一方で，一般的に DoS への防御は TEE の脅威モデルに含まれず，DoS 攻撃は Out-of-Scope と考えられることが多い [12]．Enclave 上の機密データの奪取や改ざんなど，RowHammer による DoS よりも重大な脅威が TEE に存在するかは不明である．

本研究では，RISC-V の TEE である Keystone において，RowHammer によって Mechine-mode を奪取できる可能性を示す．Machine-mode は最上位の権限であり，これが奪取されれば TEE によるコードやデータの機密性，完全性の保護は無効化になる．本研究のメインアイディアは以下の二点である．

(1) Keystone が各 Enclave の持つ権限などを管理するデータ構造である struct enclave は DRAM 上に保存されており，このデータ構造を RowHammer で書き換えれば権限奪取が可能である．
(2) Enclave 間の Context-Switching を継続的に発生させることで struct enclave に対し RowHammer によるビット反転を発生させうる．

これらのアイディアに基づき，シミュレートされた CPU とメモリの上で動作する Keystone 及び一定の条件の下で実際に Machine-mode が奪取できることを実証する．

2. 背景

2.1 Keystone

Keystone [13] は RISC-V の TEE であり，実行するコードやデータなどが第三者から読み書きされないことが保証された計算環境である Enclave を提供する．RISC-V では高い権限から順に Machine-mode, Supervisor-mode, User-mode の三種類がある．図 1 は Keystone においてそれぞれの権限で動作するプログラム群を示す．Machine-mode で実行される Security Monitor (SM) は Keystone の安全性保証の核となるプログラムであり，後述する PMP などの機能を用いて Enclave の安全性を保証する．Supervisor-mode と User-mode はそれぞれ，通常の環境であれば OS とユーザランドのプログラムが実行される権限である．Keystone では Enclave 外で Untrusted OS とアプリケーション，Enclave 内で Runtime と Enclave App (Eapp) が利用する．

SM は Physical Memory Protection (PMP) レジスタを用いて Enclave の安全性を担保する．PMP レジスタはメモリ領域の範囲とその領域に設定する権限を格納するレジスタである．権限の設定では対象領域に対し各権限（Supervisor-mode など）で許可されるアクセスの種類（読み込み，書き込み，命令フェッチ）の可否が設定できる．Keystone では SM が Enclave ごとに PMP レジスタを割り当て，Untrusted OS から Enclave へ移動する際などに PMP の設定を適切に更新する．これにより Keystone は Enclave 外のプログラムが Enclave のデータにアクセスできないことを保証する．

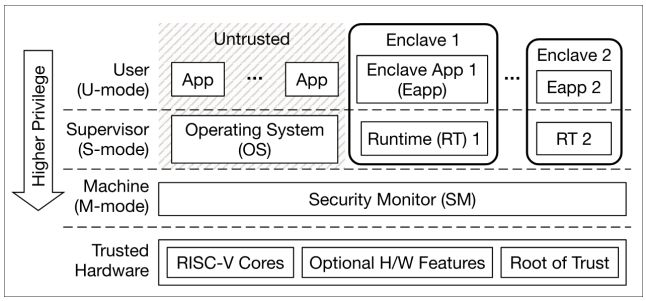

図 1: Keystone の構造 ([13] より引用)

Enclave の最大個数は，PMP レジスタの数によって制限されている．PMP は Machine-mode 専用のレジスタである pmpaddr レジスタと pmpcfg レジスタを利用して設定され，これらのレジスタは仕様上 0 個，16 個，64 個のいずれかの個数だけ実装されている．Keystone では Enclave 1 つあたり少なくとも 1 つの PMP レジスタを利用するほか，SM と Untrusted OS で計 2 つの PMP レジスタを利用するため，CPU に N 個の PMP レジスタが存在したとき Enclave は $N-2$ 個までしか作成できない．

SM は Enclave ごとにstruct enclaveという構造体を保持する．アプリケーションが Enclave での処理の開始を行うときには，図 2 のように，ecall 命令を発行することで SM に Context-Switch を依頼する．SM はstruct enclave型の配列を Enclave の個数だけ保持しており，Enclave に関する管理情報や，Context-Switch の際に退避させる必要のあるレジスタの値について保存している．SM がアプリケーションから Enclave に Context-Switch するよう要請された際には，対応するstruct enclaveに保存されている Enclave 用のレジスタの値と，レジスタにあるアプリケーションが利用した値とを入れ替える．SM は他に PMP の設定など必要な処理を行い，最終的に mret 命令を発行する．mret 命令は，表 1 の通り，mstatus レジスタの MPP フィールドの値に応じて CPU の権限レベルを変更し，mepc レジスタの示すアドレスへジャンプする．例えば，図 2 の A のようにアプリケーションから Enclave 1 へ移動する際には，まずアプリケーションがecall 命令を発行して SM へ移動する．SM は Enclave 1 に対応するstruct enclave，今回の場合は DRAM の青色の領域へアクセスし，保存されていた値とレジスタに設定する値とをスワップする．その後，mret 命令を発行して Enclave での処理を開始する．図 2 の C のように，Enclave を停止してアプリケーションへと戻る際にも同様に，レジスタの値と対応するstruct enclaveに保存された値とを入れ替え，mret 命令でアプリケーションへと戻る．

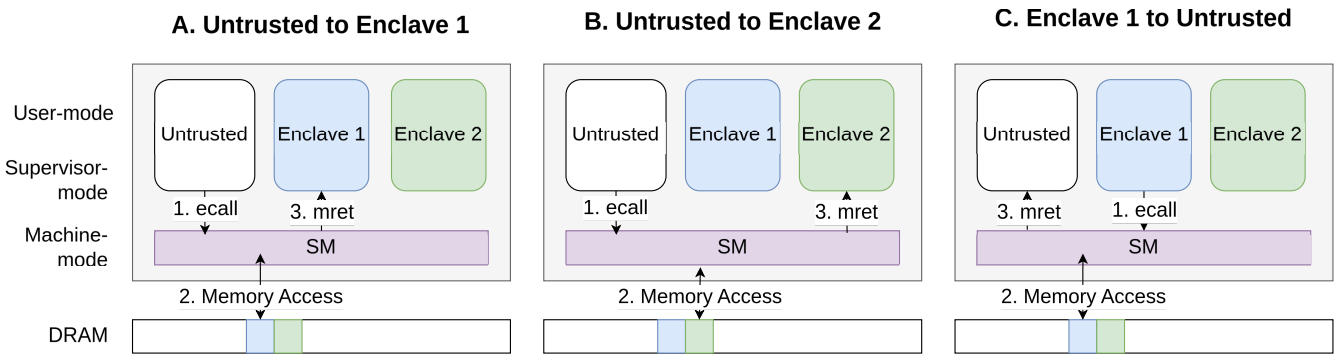

図 2: Keystone における Context-Swithing

表 1: MPP フィールドの値と権限

MPP フィールドの値	権限
00	User-mode
01	Supervisor-mode
10	Reserved
11	Machine-mode

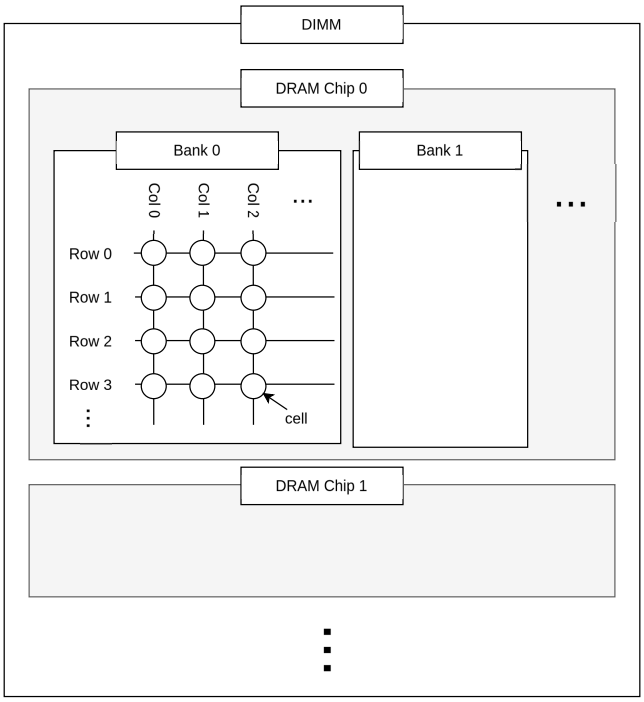

図 3: DRAM の内部構造

2.2 DRAM

現代のほぼすべてのコンピュータのメインメモリは DRAM (Dynamic Random Acceess Memory) で構成されている．図 3 に DRAM に内部構造を示す．DRAM は階層構造になっており，各 DIMM は複数の DRAM Chip を持ち，各 DRAM Chip は複数の Bank を持ちこれらが並列に動作する．各 Bank 内には 1 bit のデータを保持する cell が 格子状に集まっており，格子の一行を row，一列を column と呼ぶ．

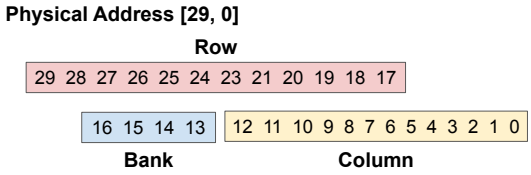

図 4: DRAM Address Mapping の一例

2.3 RowHammer

RowHammer は，DRAM のある row に 対して繰り返しアクセスすると，物理的に隣接する row のビットがフリップする場合があるという脆弱性である [6]．RowHammer は単にランダムなビットをフリップするのみならず，様々な応用攻撃が発見されている大きな脅威である．例えば Linux での権限昇格 [8]，VM への攻撃 [9]，被攻撃者プロセスへのコード注入攻撃 [10] などの応用が発見されている．

2.4 DRAM Address Mapping

DRAM Address Mapping は，物理アドレスと DRAM の各物理的要素（bank, row, column など）を対応付ける．具体的にはあるデータの物理アドレスの各ビットの値を利用し，そのデータが DRAM のどの物理的要素に保存されるかを決定する．図 4 はメモリサイズが 1 GiB の場合の DRAM Address Mapping の一例である．1 GiB のメモリは物理アドレスは 30 ビット存在する．この DRAM Address Mapping ではそのうち 29 ビット目から 17 ビット目で row を，16 ビット目から 13 ビット目で bank を，12 ビット目から 0 ビット目で column を決定する．

RowHammer を行うには，DRAM Address Mapping を知る必要がある．あるアドレスのデータを RowHammer で書き換えるには，そのデータに隣接した Row に対してアクセスを行う必要がある．しかし，攻撃対象の Row の隣接したアドレスを知るためには，DRAM Address Mapping

を元に求める必要がある．

一方で，一般的に DRAM Address Mapping についての情報は非公開である．そのため，効率的にリバースエンジニアリングする手法がいくつか提案されている [14], [15]．

3. TEE の安全性の RowHammer への耐性

一般的に，TEE は様々な攻撃に対して耐性があり，内部のデータは安全であると考えられている．Sabt ら [16] は，実行されるコードの真正性，データの完全性や機密性を保証し，ソフトウェア的な攻撃やメインメモリへの物理的攻撃へ耐性を持つものを TEE と定義している．実際，Keystone はソフトウェア的な攻撃（software adversary）のみならずハードウェア的な攻撃（hardware adversary）やサイドチャネル攻撃にも耐性を持つと主張している（文献 [13]，Table I）．このような TEE の安全性を前提とし，医療などプライバシーが重視される分野や，認証などの処理をより安全に行う技術の利用も広まっている [1], [2]．

一方で，これまで TEE の保証する安全性が RowHammer にどの程度耐性を持つかは広く研究されてこなかった．RowHammer で攻撃を行うには，ビットフリップによって攻撃者に有意な結果をもたらすデータ構造を特定し，またそのデータに対し実際にビットフリップを起こす方法を考案する必要がある．OS や VM に対する攻撃を行う研究では，ページテーブルで RowHammer を発生させることで，任意物理アドレスへの読み書きを実現し，攻撃を行う手法などが利用されている [8], [9]．一方で TEE では，ある物理アドレス以外へのアクセスを拒否する機構を用いていることが多く，また Keystone をはじめとしてページテーブルを利用しない TEE もある．このような保護機能を利用すれば RowHammer による攻撃を防ぐことができるのか，あるいはまだ脆弱なデータ構造や処理があるのかは著者らの知る限り未知の課題である．

4. 脆弱性：Machine-mode の奪取

4.1 脅威モデル

本研究では次のような脅威モデルを仮定する．条件 (2) から条件 (4) は，Keystone において Software Attacker と分類される攻撃者と同等な権限である [13]．

(1) RowHammer に脆弱な DRAM が利用されている．
(2) 攻撃者は Supervisor-mode で動作する Untrusted OS の管理者権限を持っている．
(3) 攻撃者は自由に Enclave を立ち上げることができる．
(4) 攻撃者は Machine-mode で動作する SM の操作権限や，デバイスへの物理的なアクセスを持たない．

4.2 脆弱性の構成要素

本脆弱性の構成要素の一点目は，**Enclave の権限を管理する情報がメインメモリ上に保存される**ことである．Context-Switch では，汎用レジスタの他に mstatus レジスタや mepc レジスタなどの Control and Status Registers (CSR) も入れ替える必要がある．このため Keystone ではメモリ上の struct enclave に CSR 保存用の領域を設けている．具体的には，Enclave の処理が一度停止されアプリケーションに戻るとき struct enclave の MPP フィールドには Supervisor-mode を表す 01 が格納される．00 でなく 01 になるのは，Enclave は Supervisor-mode の Runtime に一度実行を移してから停止するためである．このとき，RowHammer によって MPP フィールドの値を 1 ビットフリップさせ 11 にすることができれば，次に Enclave を起動した際に Runtime が Machine-mode で起動される．

本脆弱性の構成要素の二点目は，**struct enclave の集合が複数の row に跨って配置される**ことである．SM は複数の struct enclave を配列として保持し，その要素数は同時に起動可能な Enclave の最大個数で決まる．例えば QEMU や gem5 ではデフォルトで 16 個の PMP レジスタが利用可能なので，SM は要素数 16 のstruct enclaveの配列を持つ．1 つの struct enclave は 644 バイトと大きいため，row のサイズによっては Enclave 0-7 に対応する struct enclave と格納される row と Enclave 8 以降に対応する struct enclave は異なる row に格納される．このとき DRAM Address Mapping によってはこれらの row が隣接し，ある struct enclave に大量のアクセスを発生させれば隣接する row に格納された別の struct enclave にビットフリップが起こる可能性がある．

本脆弱性の構成要素の三点目は，**struct enclave へのアクセスを能動的に発生させられる**ことである．前述の通り Enclave と Untrusted 空間の間の Context-Switch ではレジスタ値の保存および復元のため struct enclave にアクセスする．攻撃者は Untrusted OS の Supervisor-mode の権限と任意の処理をする Enclave を起動する権限を持つため，キャッシュ追い出し手法と組み合わせることで struct enclave へのアクセスを大量発生させられる．

4.3 具体的な手順

攻撃者は RowHammer を発生させる Attacker Enclave と，権限昇格によって Machine-mode になる Target-Enclave の 2 つを利用する．攻撃の概要は図 5 の通りである．初めに攻撃者は，同一の row にstruct enclaveを持つ複数の Enclave を Attacker Enclave として確保する．また，Attacker Enclave に物理的に隣接する row にstruct enclaveを持つ Enclave を Target Enclave として確保する．次に，Attacker Enclave の開始と停止を高速に切り替えることで，Untrusted と Attacker Enclave との間を行き来し，Context-Switch を発生させる．これにより，図 5 の DRAM の赤い領域に対応する，Attacker Enclave のstruct enclaveに対して大量のアクセスが行われる．これによって RowHammer が発

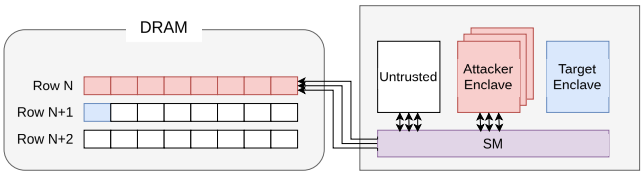

図 5: Attacker Enclave による RowHammer

表 2: ソフトウェア環境

名前	バージョン
Keystone	Master Branch (80ffb2f)
QEMU	7.2.1
gem5	24.1.0.0

生し，Target Enclave のstruct enclaveに対応する青色の領域でビットフリップが起きる．ここで，ビットフリップによって Target Enclave の MPP フィールドが 11 に書き変わった状態で起動することで，Machine-mode への権限昇格が可能である．

ここで，攻撃者はいつ Target Enclave の MPP フィールドが書き変わったかを知ることはできない．そこで，攻撃者は一定時間ごとに Target Enclave を起動し，MPP フィールドが書き変わっておらず Supervisor-mode のままであった場合には Target Enclave をまた停止させるという処理を行うことで，RowHammer が発生した瞬間が分からずとも権限昇格が可能である．

5. Proof-of-Concept 実装

ここでは，第 4 章で示した Machine-mode 奪取が一定の条件の下で実際に可能なことを示すための実装を示す．

5.1 ソフトウェア環境

RowHammer を実機で行うには，一般的に公開されていない DRAM Address Mapping の情報を入手する必要があり，攻撃手法の本質とは異なる部分での作業が必要となる．そこで本論文では，脆弱性の発現可能性や影響の議論を優先するため，gem5 [17] 上で RowHammer の影響を再現できる Hamulator [18] を利用する．また，RowHammer の再現が必要ない実験については，QEMU [19] を利用することで高速化を図る．QEMU と gem5 のバージョンは表 2 の，gem5 の設定は表 3 の通りである．また，DRAM Address Mapping は下位ビットから順に column が 10 bit，row が 16 bit，bank group が 2 bit，bank が 2 bit，rank が 1 bit という設定を用いる．

Keystone の実装は，GitHub[*1]にて入手可能である．我々は，GitHub の/docker 以下にある Dockerfile をもとに実装を行う．この環境では struct enclaveのベースアドレスは 0x80060b00，サイズは 644 バイトである．

[*1] https://github.com/keystone-enclave/keystone

表 3: gem5 の設定

項目	名前	パラメータ
CPU	TimingSimpleCPU	10 cores
DRAM	SimgleChannelDDR4_2400	8 GiB
Cache	PrivateL1CacheHierarchy	128 KiB
Board	RiscvBoard	1 GHz

5.2 Cache Flush

RowHammer を起こすには，メモリアクセスの後に Cache Flush を行ってキャッシュの内容を DRAM に書き戻す必要がある．RISC-V では Zicbom 拡張にて定義される cbo.flush 命令を利用することで特定のアドレスに対応するキャッシュの内容を書き戻すことができる．しかし，PMP レジスタでアクセスが制限されている領域に対してフラッシュ命令を発行することはできないので，攻撃者は SM の領域にあるstruct enclaveに対して cbo.flush を行うことはできない．

そこで，cbo.flush 命令を利用する代わりに，適当なサイズのデータにアクセスしてキャッシュを溢れされることでキャッシュフラッシュを行う．第 5.1 章の通り，今回の環境には 128 KiB の 8 ウェイセットアソシアティブ方式の L1 D-cache が存在している．ある Enclave に対応する struct enclave が格納される set index はそのアドレスからできるため, 同一セットを共有する U ntrusted OS 上のデータに対して 8 回アクセスを行うことで struct enclaveの内容を DRAM に書き戻す．

5.3 権限昇格の検知

RowHammer によっていつstruct enclaveが書き変わったかを検知する方法は無いため，定期的に Target Enclave を立ち上げて，Machine-mode へと権限昇格できたか確認する．しかし RISC-V において現在の CPU の実行権限を確認する方法は無い．そこで Supervisor-mode から mstatus などの Machine-mode 用のレジスタへアクセスを試みると Illigal Instruction 例外が発生することを利用する．Keystone において Supervisor-mode で発生した Illigal Instruction 例外は，最終的に stvec で指定される Supervisor-mode のトラップハンドラで処理される．従ってハンドラを適切に設定した状態で Machine-mode 専用のレジスタへアクセスを試み，トラップハンドラへ処理が飛べば Supervisor-mode，飛ばなければ Machine-mode であると判定できる．

5.4 SM の変更

Keystone の実装において，仕様と異なっている部分があったため修正する．もともと Keystone は Enclave 1 つあたり最小で 1 つの PMP レジスタがあれば動作し，CPU に N 個の PMP レジスタがあれば $N-2$ 個の Enclave を

立ち上げることが可能であると明記されている [13]．一方で GitHub 上の SM の実装では，Enclave のメモリ領域以外に共有メモリ領域の確保を行うため，Enclave 1 つあたり少なくとも 2 つの PMP レジスタを利用している．共有メモリ領域を確保しなくとも Enclave は正常に動作するため，共有メモリ領域のために PMP レジスタを使わないよう SM を変更する．

6. 評価実験

6.1 Context-Switch による RowHammer

ここでは，複数の Enclave で Context-Switch を起こし続けることでstruct enclaveで RowHammer が発生することを確認する．Hammulator は，それぞれの row へのアクセス回数を hammer_count に保存する．Hammulator がある閾値，今回の実験環境では 10,000 を越えると，RowHammer が起きたとして適当なビットをフリップさせる．struct enclaveに対応する hammer_count の値を出力するように Hammulator を改造して実験を行う．

struct enclaveへのアクセス回数を可能な限り増やすため，複数の Enclave で同時に Context-Switch を発生させる必要がある．今回の設定では 0 から 7 番目までの Enclave に対応するstruct enclaveが DRAM の同一の row に存在しているので，この 8 つの Enclave で Context-Switch を起こし続けた．

実験の結果，シミュレーション時間 1.8 秒，実時間 5 時間 3 分 5 秒後にstruct enclaveに RowHammer によるビットフリップが発生することを確認した．よって，Context-Switch を起こし続けることで RowHammer に必要なだけのアクセスが十分行われることが分かった．また実験中に確認できた hammer_count の最大値は閾値を大きく越える 186,028 であった．なお hammer_count は DRAM 内の電気的動作によって一定時間ごとにリセットされることに注意が必要である．

6.2 MPP の書き換えによる権限昇格

ここでは，struct enclaveの MPP フィールドが書き変わることで Machine-mode へ権限昇格されることを確認する．また，5.3 章で実装した，CPU の現在の権限を確認する関数が正しく動作することを確認する．

QEMU 上において，次の実験を行う．

(1) SM のコードは変えずに Enclave を起動し，Enclave から mstatus レジスタへアクセスを試みる
(2) struct enclaveの MPP フィールドの値が常に 11 になるよう SM を書き換えた上で Enclave を起動し，mstatus レジスタへアクセスを試みる
(3) (1), (2) においてそれぞれ，5.3 章で実装した関数が権限を適切に認識できるか確認する

もし Enclave が Machine-mode で起動すれば，Machine-mode でのみアクセスが可能な mstatus レジスタへアクセスできるはずである．一方で，もし Enclave が Supervisor-mode で起動すれば，mstatus レジスタへのアクセスは Illigal Instruction 例外を発生させ，アクセスが拒否される．

表 4: MPP フィールドの値と実行権限

MPP フィールドの値	mstatus レジスタへのアクセス	5.3 章のプログラムの判定
01	失敗	Supervisor-mode
11	成功	Machine-mode

結果は，表 4 の通りである．(1) ではレジスタへのアクセスは拒否され，処理が停止された．一方で (2) ではレジスタへのアクセスに成功した．そこで，struct enclaveの値が 11 になれば Machine-mode へ権限昇格できることが確認できた．(3) については，(1), (2) をそれぞれ Supervisor-mode, Machine-mode と適切に判定した上で処理を再開することに成功し，現在の CPU の実行権限を適切に判定できることを確認した．

6.3 End-to-End の検証

ここでは，実際にシミュレートされたマシンの起動から Machine-mode の奪取までの一連の流れを実証する．ただし今回の実験では hammer_count が閾値を越え RowHammer が発生した際には struct enclave の MPP フィールドがビットフリップするように固定した．実際のマシンでは row 中のフリップするビットは事前に分からないため，狙ったビットがフリップする row の上に struct enclave を配置する何らかの手法が必要になる．既存研究では狙ったビットがフリップする row に攻撃対象データを配置することを Memory Massaging と呼ぶ．

実証の手順は以下の通りである．

(1) Attacker Enclave を 0.01 秒ごとに 1 つづつ，合計 8 つ初期化する
(2) 0.5 秒待機する
(3) Target Enclave を 1 つ初期化する
(4) Target Enclave を一度起動したあと，Linux 空間で一時停止させる
(5) 全ての Attacker Enclave で Context-Switch を発生させ続ける
(6) 定期的に Target Enclave を起動し，Machine-mode になっているか確認する

手順 (4) の意図は，Target Enclave の管理情報を保持する struct enclave を初期化することである．ある Enclave に対応する struct enclave はその Enclave の初回起動時に初期化されるため，この手順がないと RowHammer 攻撃でフリップした値を初期値で上書きしてしまう．

図 6 は上記手順を実行した際の gem5 のコンソール出力である．出力の 41 行目は Target Enclave が Machine-

```
01: Start Attacker Enclave Execution 0
02: Start Attacker Enclave Execution 1
03: Start Attacker Enclave Execution 2
04: Start Attacker Enclave Execution 3
05: Start Attacker Enclave Execution 4
06: Start Attacker Enclave Execution 5
07: Start Attacker Enclave Execution 6
08: Start Attacker Enclave Execution 7
09: Start Victim Enclave Execution
10: [+] Executing victim enclave
11: [-] We are in supervisor mode now
12: [+] First execution completed
13: [    1.545495] l1d_evictor: initialized with
line=64 sets=256 ways=8, buffer_size=212992
14: [    1.545561] [+] Victim is ready
15: [+] Executing victim enclave
16: [-] We are in supervisor mode now
17: [+] Executing victim enclave
18: [-] We are in supervisor mode now
19: [+] First execution completed
20: [+] Executing victim enclave
21: [-] We are in supervisor mode now
22: [+] Executing victim enclave
23: [-] We are in supervisor mode now
24: [    1.681595] [+] 5000 Context switches
25: [+] Executing victim enclave
26: [-] We are in supervisor mode now
27: [    1.719208] [+] 10000 Context switches
28: [+] First execution completed
29: [    1.756880] [+] 15000 Context switches
30: [+] Executing victim enclave
31: [-] We are in supervisor mode now
32: [    1.778730] [+] 20000 Context switches
33: [    1.800473] [+] 25000 Context switches
34: [    1.822209] [+] 30000 Context switches
35: [+] Executing victim enclave
36: [-] We are in supervisor mode now
37: [    1.843959] [+] 35000 Context switches
38: [+] First execution completed
39: [    1.865697] [+] 40000 Context switches
40: [+] Executing victim enclave
41: [+] Congratulations! We are in machine-mode now
42: [+] mstatus = a00000080
```

図 6: End-to-Endo の検証における gem5 の出力

mode になったことを検知した際の出力であり，実際に Machine-mode が奪取できることが確認できた．1-8 行目では，Attacker Enclave の起動処理を開始している．9 行目では Target Enclave の起動処理を行い，その後は 10-11，15-16 行目など定期的に起動して Target Enclave の実行権限を確認している．12，19，28 行目などでは，Attacker Enclave の起動が完了して Context-Switch を発生させ始めている．13 行目は 5.2 章で実装した Cache Flush の初期化が終わったことを示す．24，27，29 行目のように順調に Context-Switch を行なった回数が増え，最終的に，Linux の起動から 1.9 秒後，40000 回目の Context-Switch が完了した後に Target Enclave が Machine-mode で起動できた．

7. 議論

7.1 実際のハードウェアでの脆弱性

本研究では Machine-mode の奪取の**可能性**を示したに留まり，実際のハードウェアで脆弱性が発現するかは未知である．具体的なチャレンジとして，DRAM Address Mapping への依存性がある．実際のハードウェアの DRAM Address Mapping は CPU やマシンのメモリ容量ごとに異なる．DRAM Address Mapping に依存せず struct enclave への RowHammer を実現する Memory Massaging 手法が必要である．

7.2 脆弱性の緩和手法

本研究で指摘した脆弱性は以下のような手法により緩和できる可能性がある．

(1) MPP フィールドの拡張：現在の MPP フィールドは 2 ビットの値を保持する．これを拡張し 1 ビット（以上）の冗長性を持たせ，例えば Machine-mode に対応する値を 111 にすることが考えられる．RowHammer で特定の 2 ビットをフリップすることは 1 ビットをフリップする場合に比べ非常に困難であるため，これにより本脆弱性を緩和できると考えられる．

(2) struct enclave の CPU 内への保存：本脆弱性の本質は struct enclave が DRAM に保存されていることである．従って struct enclave を CPU 内の SRAM 領域に保持すれば本脆弱性は根本から不成立になる．struct enclave の個数は最大でも 64 個であるため，それらを全て保存する SRAM 領域を用意することは大きなオーバーヘッドではないと考えられる．

8. 関連研究

SGX-Bomb [11] では，メモリ完全性保証のある TEE における RowHammer による攻撃手法が示された．Intel SGX はメモリアクセス時に完全性を検証し，検証に失敗すればプロセッサをロックダウンする．そのため，RowHammer によって意図的にメモリを改変することで，Denial-of-Service を行うことが可能であった．一方で，Keystone や TrustZone など近年の多くの TEE は完全性保証行なっておらず，そのような TEE に対して SGX-Bomb で攻撃することはできない．また，SGX-Bomb では DoS しか行えず，Enclave 内のデータの読み書きは不可能である．

SgxPectre [20] では，Intel SGX Enclave において，Specture と呼ばれる CPU の脆弱性を用いた攻撃手法を示した．SgxPecture は，分岐予測器が使用する Buffer を他プロセスから汚染することで投機的実行によるメモリアクセスを引き起こし，キャッシュに機密情報を乗せる．その後，キャッシュサイドチャネル攻撃によって Enclave 内の機密情報を窃取できる．

Foreshadow [21] では，Intel SGX Enclave において，Meltdown と呼ばれる CPU の脆弱性を用いた攻撃手法である．Foreshadow は、Enclave 内のメモリに対して Enclave 外からアクセスを行うとページフォルトが発生し、ページフォルト前に投機的実行が発生する。この投機的実行において Enclave 外からアクセスしたアドレスに対応する値がキャッシュに乗る。その後、キャッシュサイドチャネル攻撃によって Enclave 内のデータを窃取できる。しかし、

SgxPecture と同様に Enclave 内の情報は窃取できるが，Enclave の領域へ書き込みを行うことはできない．

既存研究に対する本研究の違いは，本研究が Machine-mode の奪取を通じて Enclave 領域の書き込みをも実現しうること，読み込みに関しても特権奪取後は任意のメモリ領域を読み込めるためサイドチャネル攻撃を利用した読み込みより高速にデータを搾取できることである．

9. おわりに

TEE はプライバシーやセキュリティの重要な分野で利活用が進んでいる一方，これまで TEE が RowHammer に対して耐性があるのかは知られていなかった．本研究では RISC-V Keystone TEE において，RowHammer によって Machine-mode の奪取が可能な可能性を明らかにした．今後は本脆弱性の実機での確認および緩和手法の実装方法の検討や評価を行う．

謝辞 本研究は，JST 次世代科学技術チャレンジプログラムおよび JST，さきがけ，JPMJPR22P1 の支援を受けたものである．

参考文献

[1] Li, J., Luo, X. and Lei, H.: TrustHealth: Enhancing eHealth Security with Blockchain and Trusted Execution Environments, *Electronics*, Vol. 13, No. 12 (2024).

[2] Zhang, Y., Zhao, S., Qin, Y., Yang, B. and Feng, D.: TrustTokenF: A Generic Security Framework for Mobile Two-Factor Authentication Using TrustZone, *IEEE Trustcom/BigDataSE/ISPA*, pp. 41–48 (2015).

[3] Ayoade, G., Karande, V., Khan, L. and Hamlen, K.: Decentralized IoT Data Management Using BlockChain and Trusted Execution Environment, *International Conference on Information Reuse and Integration (IRI)*, pp. 15–22 (2018).

[4] Chen, Y., Luo, F., Li, T., Xiang, T., Liu, Z. and Li, J.: A Training-Integrity Privacy-Preserving Federated Learning Scheme with Trusted Execution Environment, Vol. 522, pp. 69–79.

[5] Mo, F., Shamsabadi, A. S., Katevas, K., Demetriou, S., Leontiadis, I., Cavallaro, A. and Haddadi, H.: DarkneTZ: towards model privacy at the edge using trusted execution environments, *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, MobiSys '20, p. 161–174 (2020).

[6] Kim, Y., Daly, R., Kim, J., Fallin, C., Lee, J. H., Lee, D., Wilkerson, C., Lai, K. and Mutlu, O.: Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors, *International Symposium on Computer Architecuture (ISCA)*, p. 361–372 (2014).

[7] Jattke, P., Wipfli, M., Solt, F., Marazzi, M., Bölcskei, M. and Razavi, K.: ZenHammer: Rowhammer Attacks on AMD Zen-based Platforms, *USENIX Security Symposium*, pp. 1615–1633 (2024).

[8] Mark, S. and Thomas, D.: Exploiting the DRAM rowhammer bug to gain kernel privileges, (online), available from ⟨`https://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html`⟩ (accessed 2025-08-13).

[9] Razavi, K., Gras, B., Bosman, E., Preneel, B., Giuffrida, C. and Bos, H.: Flip Feng Shui: Hammering a Needle in the Software Stack, *USENIX Security Symposium*, pp. 1–18 (2016).

[10] Yoshioka, K. and Akiyama, S.: GbHammer: Malicious Inter-process Page Sharing by Hammering Global Bits in Page Table Entries, *Fourth Workshop on DRAM Security (DRAMSec)*, pp. 1–7 (2024).

[11] Jang, Y., Lee, J., Lee, S. and Kim, T.: SGX-Bomb: Locking Down the Processor via Rowhammer Attack, *Workshop on System Software for Trusted Execution (SysTex)*, pp. 1–6 (2017).

[12] Nilsson, A., Bideh, P. N. and Brorsson, J.: A Survey of Published Attacks on Intel SGX (2020).

[13] Lee, D., Kohlbrenner, D., Shinde, S., Asanović, K. and Song, D.: Keystone: an open framework for architecting trusted execution environments, *European Conference on Computer Systems (EurSys)*, pp. 1–16 (2020).

[14] Wang, M., Zhang, Z., Cheng, Y. and Nepal, S.: DRAMDig: a knowledge-assisted tool to uncover DRAM address mapping, *Design Automation Conference (DAC)*, pp. 1–6 (2020).

[15] Pessl, P., Gruss, D., Maurice, C., Schwarz, M. and Mangard, S.: DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks, *USENIX Security Symposium*, pp. 565–581 (2016).

[16] Sabt, M., Achemlal, M. and Bouabdallah, A.: Trusted Execution Environment: What It is, and What It is Not, *2015 IEEE Trustcom/BigDataSE/ISPA*, pp. 57–64 (2015).

[17] Lowe-Power, J., Ahmad, A. M., Akram, A., Alian, M., Amslinger, R., Andreozzi, M., Armejach, A., Asmussen, N., Bharadwaj, S., Black, G., Bloom, G., Bruce, B. R., Carvalho, D. R., Castrillón, J., Chen, L., Derumigny, N., Diestelhorst, S., Elsasser, W., Fariborz, M., Farahani, A. F., Fotouhi, P., Gambord, R., Gandhi, J., Gope, D., Grass, T., Hanindhito, B., Hansson, A., Haria, S., Harris, A., Hayes, T., Herrera, A., Horsnell, M., Jafri, S. A. R., Jagtap, R., Jang, H., Jeyapaul, R., Jones, T. M., Jung, M., Kannoth, S., Khaleghzadeh, H., Kodama, Y., Krishna, T., Marinelli, T., Menard, C., Mondelli, A., Mück, T., Naji, O., Nathella, K., Nguyen, H., Nikoleris, N., Olson, L. E., Orr, M. S., Pham, B., Prieto, P., Reddy, T., Roelke, A., Samani, M., Sandberg, A., Setoain, J., Shingarov, B., Sinclair, M. D., Ta, T., Thakur, R., Travaglini, G., Upton, M., Vaish, N., Vougioukas, I., Wang, Z., Wehn, N., Weis, C., Wood, D. A., Yoon, H. and Zulian, É. F.: The gem5 Simulator: Version 20.0+, *arXiv:2007.03152*, pp. 1–20 (2020).

[18] Thomas, F., Gerlach, L. and Schwarz, M.: Hammulator: Simulate Now – Exploit Later, *Workshop on DRAM Security (DRAMSec)*, pp. 1–7 (2023).

[19] Bellard, F.: QEMU, a fast and portable dynamic translator, *USENIX Annual Technical Conference (ATC)*, pp. 41–46 (2005).

[20] Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z. and Lai, T. H.: SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution, *2019 IEEE European Symposium on Security and Privacy (EuroSP)*, pp. 142–157 (2019).

[21] Bulck, J. V., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y. and Strackx, R.: Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution, *USENIX Security Symposium*, pp. 991–1008 (2018).