B-002

Enclave に対するデータ競合攻撃を防止する検査コードの生成

古都 瀬菜[1)]　畑 輝史[1)]　河野 健二[1)]
Sena furuichi　Terufumi Hata　Kenji Kono

1　はじめに

悪意のあるオペレーティングシステム (OS) が Enclave に対して行う攻撃のひとつにデータ競合を引き起こす方法がある．Enclave 内の並行性制御に関する脆弱性を悪用し，データ競合を起こすように OS が意図的にスレッドをスケジューリングする．本論文では，OS によるデータ競合攻撃を防ぐために Enclave 内のコード実行順序を制限する手法を提案する．アプリケーション開発者の意図しない順序でコードが実行されることのないよう，コードの実行順序を明示的に指定できるようにする．この制限により並行性制御の脆弱性のあるコードでも脆弱性を悪用できなくなることを確認した．

2　背景

2.1　TEE

TEE (Trusted Execution Environment) は隔離され信頼性のある実行環境を提供し，データやコードを他のソフトウェアから保護する [1]．この隔離された信頼性のある実行環境のことを Enclave と呼ぶ．TEE は高度なプライバシーを実現するためのベースとして利用する機能である．OS や Hypervisor などの特権ソフトウェアに悪意があっても，データのプライバシーを守れることが強みである．代表的な TEE のうちの一つに Intel SGX [2] がある．

2.2　Enclave 内のソフトウェアの脆弱性

SGX (Software Guard Extensions) が対策できる攻撃には限りがある．暗号的に保護できる攻撃に関しては対策することができ，プロセス内攻撃や OS・Hypervisor による攻撃，CPU 以外のハードウェアへの攻撃は SGX によって防ぐことができる．

一方で，サイドチャネル攻撃に関しては脆弱である．このように SGX を使用していても必ずしも安全とは限らない．Enclave 内で動作するソフトウェアにバグがあると脆弱性に繋がる．OS や Hypervisor などの特権ソフトウェアが Enclave 内のコードの実行の流れを変えてしまったり，Enclave から特権ソフトウェアにデータが流出してしまったりする危険性がある．

2.3　データ競合による脆弱性

2.2 章で説明したように，Encalve 内のソフトウェアに脆弱性があると危険なことが分かる．Encalve 内で動作するソフトウェアの脆弱性の一つにデータ競合による脆弱性がある [3]．データ競合とは，複数のスレッドが同期化されない状態で共有メモリにアクセスし，少なくとも一方が書き込み操作を行う場合に発生する競合状況のことである．このような問題は，プログラムの予測不可能な動作やセキュリティ違反を引き起こす可能性がある．データ競合によって引き起こされる脆弱性は，複数スレッドが同じメモリをアクセスする際に生じ，メモリが意図しないタイミングで変更されることでプログラムの中身が変化してしまう危険性がある．関数の呼び出し順序を制限しないことなどが原因でデータ競合は生じる．

SGX のような TEE は，機密データを保護するための隔離された Enclave 環境を提供する．この環境では外部のシステムソフトウェアからの攻撃を防ぎながら，信頼されたコードを実行することができる．しかし 2.2 章で説明したように，SGX で防ぐことのできる攻撃には制約があり，データ競合は大きな脅威になる．

通常、データ競合は非決定的に発生するため再現が困難で脆弱性の発見が難しい．しかし，SGX 環境では攻撃者が OS を制御できる場合，競合の発生を意図的かつ決定的に操作することが可能になる．このように，データ競合を意図的に引き起こすことで攻撃を仕掛けることができる．

例えば，Enclave 内で使用される共有変数に対する保護が不十分な場合，攻撃者はその変数の値を変更し，結果としてコードの制御フローを操作したり，機密データを漏洩させることができる．

2.4　脅威モデル

本論文では，データ競合による脆弱性を利用した攻撃を防ぐことを目指す．攻撃者は OS 完全に制御できるとき，データ競合を意図的に引き起こしこのような攻撃を仕掛けることができる．また，攻撃者は Enclave 内で発生するスレッドのスケジューリングや例外処理を細かく操作し，特定のタイミングで共有変数にアクセスすることでデータ競合を引き起こすことで攻撃を仕掛ける．以上の点を踏まえ，想定している脅威モデルに関してまとめると次のようになる．攻撃者は OS を制御することが可能であり，スケジューリングのタイミング等を自由に制御できる．また，Enclave 内で実行されるコードの詳細を把握している．また，Enclave 内で動作するコードは複数のスレッドによる実行が可能である様に設定されており，データ競合による脆弱性が含まれている．

3　アプローチ

本論文では，OS によるデータ競合攻撃を防ぐために Enclave 内の関数呼び出し順序を制限する．これにより OS による Enclave の関数呼び出し順序を無視した実行を防ぐ．そのために Enclave のアプリケーションの開発者が Enclave の関数呼び出し順序を明示的に決定し，関数呼び出し順序を示すオートマトンをソースコードのコンパイル時に与えると，関数呼び出し順序を制限するコードが Enclave 内のコードに生成される．

この提案手法の利点を説明する．まず，関数実行時に関数呼び出し順序を確認するため，動的に脆弱性を防ぐことが可能である．また，開発者が関数呼び出し順序を自由に定義できるため，アプリケーション固有の要件に対応することが可能である．さらに，検査コードの実行は軽量で，パフォーマンスへの影響を最小限に抑えることが期待できる．

本論文では，Intel SGX を用いて実装を行う．Enclave 内のコードは sgx_edger8r によって自動で生成され

1)　慶應義塾大学 keio university

る．この sgx_edger8r に変更を加えることで検査コードの生成を実現する．提案手法を実装するために，sgx_edger8r に約 200 行の変更を加えた．

4 実験

今回実装した検査コードに関して，検査コードがある場合とない場合で実行時間に差が出るかどうか調べるために実験を行う．検査コードがある場合，ない場合それぞれの時に，ecall で呼ばれた関数の中身がある場合とない場合の計 4 パターンを比較する．ecall で呼ばれた関数の中身が多いほど検査コードの影響は少なくなるため，この 4 パターンで測定を行う．

4.1 実験環境

測定に使用した環境を表 1 に示す．

CPU	Intel(R) Xeon(R) Gold 6330 CPU @ 2.00 GHz
コア数	56
OS	Ubuntu 20.04.6 LTS (Linux Kernel 5.4.0-200-generic)

表 1: 実験環境

4.2 実験結果

実験の測定結果を示す．ecall で呼ばれた関数の中身がある場合は，実際に検出されたデータ競合の危険がある関数を用いて実験を行った．このとき関数呼び出し順序を満たしていて，エラーは生じずに関数は適切に実行されるとする．ecall に入る直前から，ecall から return を受け取る直後までの実行時間を 10000 回測定した時の平均時間を求めた．計測されたサイクル数を Base Frequency で割り算することで実行時間を計測すると，図 1 のようになる．4 パターンの実行時間に差はなく，検査コードの有無は実行時間に影響を与えないことがわかる．すなわち，提案手法によって実行時間に影響を与えることなく，データ競合を防ぐための関数呼び出し順序を確認することができる．

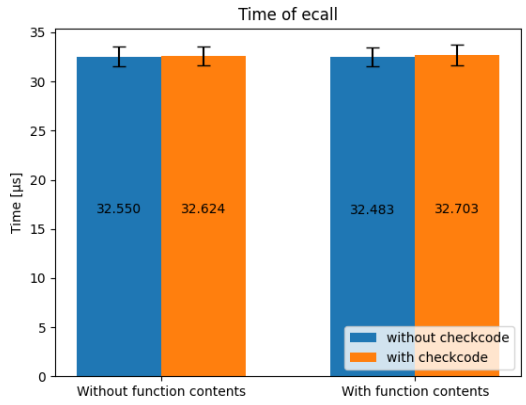

図 1: ecall の実行時間を 10000 回測定した時の平均時間

5 関連研究

5.1 SGXRACER

SGXRACER[3] では，Intel SGX のような TEE を対象とした新たな攻撃手法 Controlled Data Race Attacks を提案し、それに対抗するための検出ツール “SGXRACER” を紹介している．

5.2 Typestate

Typestate[4] では，“型” という概念を改良した新しいプログラミング言語の概念である Typestate を提案している．従来のデータオブジェクトの型はそのオブジェクトに許される操作の集合を決定するのに対し，Typestate は特定の文脈において許される操作の部分集合を決定する．この Typestate を追跡する Typestate checking はプログラムの信頼性を向上させるプログラム解析技術であり，構文的には合法であるが意味的には未定義な実行シーケンスをコンパイル時に検出する．

6 まとめ

近年のクラウド環境では，OS や Hypervisor が攻撃を仕掛ける危険性があり，アプリケーションをこれらから保護する必要性が高まっている．このような課題に対処する技術として、TEE が登場した．しかし、TEE を使用すると確実に安全が保障されるとは限らない．Enclave 内の脆弱性を利用した攻撃の 1 つとしてデータ競合による攻撃というものがある．このデータ競合による脆弱性を悪用することで，攻撃者は機密データの漏洩や処理の改竄を実現することができる．

本論文では、このようなデータ競合による攻撃を防ぐための検査コードを提案・実装し，その有効性を実証した．本論文で実装した検査コードを適用することで，意図的に発生させたデータ競合がすべて防がれることを確認できた．これにより，関数呼び出し順序の制限によってデータ競合を効果的に防止できることが実証された．結果，提案手法によって実行時間に影響を与えることなく，データ競合を防ぐための関数呼び出し順序を確認することができることがわかった．

謝辞

本研究は，経済安全保障重要技術育成プログラム（K Program），【JPMJKP24U4】の支援を受けた．

参考文献

[1]Stavros Volos and Kapil Vaswani and Rodrigo Bruno, Graviton: Trusted Execution Environments on GPUs, *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 681-696, Carlsbad, CA, USENIX Association, October 2018.

[2]Intel, Intel software guard extensions (intel sgx), https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/software-guard-extensions.html, (Accessed on 01/17/2025).

[3]Sanchuan Chen, Zhiqiang Lin, and Yinqian Zhang. Controlled data races in enclaves: attacks and detection. *In 32nd USENIX Security Symposium (USENIX Security 23)*, pages 4069–4086, Anaheim, CA. USENIX Association, August 2023.

[4]Robert E. Strom and Shaula Yemini. Typestate: a programming language con- cept for enhancing software reliability. *IEEE Transactions on Software Engineering*, pages 157–171, 1986.