TCS Security Analysis in Intel SGX Enclave MultiThreading

Tong Zhang[1], Xiangjie Cui[1], Yichuan Wang[1,2,*], Yanning Du[1], Wen Gao[1]
[1]School of computer science and engineering, Xi'an University of technology, Xi'an, China
[2]Shaanxi Key Laboratory for Network Computing and Security Technology
Email: chuan@xaut.edu.cn

2022 International Conference on Networking and Network Applications (NaNA) | 978-1-6654-6131-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/NANA56854.2022.00054

***Abstract*—With the rapid development of Internet Technology in recent years, the demand for security support for complex applications is becoming stronger and stronger. Intel Software Guard Extensions (Intel SGX) is created as an extension of Intel Systems to enhance software security. Intel SGX allows application developers to create so-called enclave. Sensitive application code and data are encapsulated in Trusted Execution Environment (TEE) by enclave. TEE is completely isolated from other applications, operating systems, and administrative programs. Enclave is the core structure of Intel SGX Technology. Enclave supports multi-threading. Thread Control Structure (TCS) stores special information for restoring enclave threads when entering or exiting enclave. Each execution thread in enclave is associated with a TCS. This paper analyzes and verifies the possible security risks of enclave under concurrent conditions. It is found that in the case of multithread concurrency, a single enclave cannot resist flooding attacks, and related threads also throw TCS exception codes.**

***Index Terms*—Intel SGX, Enclave, Thread Control Structure, Multithread Concurrency, Flood Attacks**

I. Introduction

Intel SGX are a set of extensions to Intel Architecture. It is designed to provide integrity and confidentiality guarantees for security-sensitive computations performed on computers where all privileged software (kernel, hypervisor, etc.) may be malicious [1]. Intel launched the SGX instruction set extension, which is guaranteed by hardware security and does not depend on the security status of firmware and software. Intel SGX provides a TEE in user space, and implements different programs and access control through a new set of instruction set extensions mechanism. It runs in isolation between users to ensure the confidentiality and integrity of users' critical code and data from being damaged by malware. Unlike other security technologies, Intel SGX's Trusted Computing Base (TCB) consists of hardware only. The defects of software security loopholes and threats existing in the software based TCB itself are avoided. The system security is greatly improved. In addition, Intel SGX can ensure a TEE at runtime, malicious code cannot access and tamper with the protection content of other programs during runtime. The security of the system is further enhanced. The extension based on the instruction set and the independent authentication method make the application program flexibly call this safety feature is verified [2].

Intel SGX allows application developers to create so-called enclave. Sensitive application code and data are encapsulated in TEE by enclave. TEE is completely isolated from other applications, operating systems, and administrative programs [3]. Enclave is a protected content container for storing application sensitive data and code. Intel SGX allows applications to specify the code and data parts that need to be protected.

As the core fabric of Intel SGX, Encalve has been the focus of various attacks. This paper analyzes and demonstrates the security risks of Enclave in the case of multi-thread concurrency. The rest of this paper is organized as follows. The first part introduces the related principles of Intel SGX technology and focuses on the Enclave structure. The second part introduces the attacks against the Intel SGX Enclave and the transformation of the enclave. The third part introduces the hidden dangers of multi-threaded concurrency security of enclave in Intel SGX. The fourth part makes a theoretical analysis of the hidden dangers of the multi-thread concurrency security of enclave in Intel SGX. The fifth part designs experiments and verifies the hidden dangers of multi-thread concurrency security in enclave. The sixth part is the summary of the work.

II. Related Work

In the analysis and research of the TEE, there are transformations and applications based on a single enclave. For example, Wang Wenhao et al. proposed the use of Liveries to achieve the ability to safely run different user tasks within a single enclave [4]. This article will analyze and study the security issues that exist when a single enclave executes in multithread concurrency.

To describe how the enclave trades off the available parallelism for a set of parallelized workloads on Intel SGX capable multi-core CPUs. Brandon D'Agostino and Omer Khan developed a microbenchmark to study the impact of threads as a function of application characteristics [5]. This article will analyze and study the concurrent execution of enclave by multiple logical CPUs in the subsequent parts.

ShenYouren et al. propose to implement safe and efficient multitasking within an enclave of Intel SGX, introducing a library operating system (LibOS) into Intel SGX so that legacy applications can run within the enclave with little or no modification [6].

Nico Weichbrodt et al. propose that it is easier to exploit synchronization errors in Intel SGX enclaves compared to traditional applications. This is because, by design, the attacker

978-1-6654-6131-3/22/$31.00 ©2022 IEEE
DOI 10.1109/NaNA56854.2022.00054

can control the thread scheduling of the enclave in the Intel SGX attacker model [7].

For the migration of applications located in different TEEs (such as Intel SGX, AMD SEV and ARM TrustZone), Gu, Jin-Yu et al. propose UniTEE, which uses a microkernel-based design in one of the above three hardware TEEs. It provides a unified enclave programming abstraction, and realizes safe enclave migration by integrating heterogeneous migration technology [8].

T. Yavuz et al presents ENCIDER, an accurate side-channel analysis tool that detects timing and cache side-channel vulnerabilities in Intel SGX applications by inferring potential timing watchpoints and incorporating the Intel SGX programming model into the analysis. ENCIDER uses dynamic symbolic execution, decomposes side-channel requirements based on the bounded non-interference property, and implements byte-level information flow tracking through API modeling [9].

Kockan, Can, et al. introduce SkSES, a hardware–software hybrid approach for privacy-preserving collaborative GWAS, which improves the running time of the most advanced cryptographic protocols by two orders of magnitude. The SkSES approach is based on TEEs offered by current-generation microprocessors in particular, Intel's SGX [10].

E. Antolak and A. Pułka presents balanced heuristic techniques of static tasks scheduling in multi-core real-time system architecture. The main objective was to minimize the energy consumed by the system without causing deadlines to be missed [11].

From the perspective of trusted computing, Xinhong Hei et al. propose and implement a trusted static measurement method of the Android system based on TrustZone to protect the integrity of the system layer and provide a trusted underlying environment for the detection of the Android application layer [12].

A trusted android device security communication method based on TrustZone is proposed by Yichuan Wang et al. It constructs a secure communication between mobile devices without a trusted third party [13].

III. Security risks of multithreading concurrency in enclave of Intel SGX

A. An overview of the life cycle of an enclave

The life cycle of enclave involves a total of seven instructions such as ECREATE, EADD, EEXTEND, INITIALIZATION, EENTER, EEXIT, EREMOVE, etc. The specific details are described in Fig 1.

B. Enclave's threading mechanism

The Intel SGX design fully supports multi-core processors. Each logical processor can execute code from the same enclave through a different thread. The Intel SGX implementation uses a TCS for each logical processor executing enclave code. Each TCS is stored in a dedicated Enclave Page Cache (EPC) page with an EPCM entry of type PT-TCS. EPC pages with TCS are not directly accessible, even the code of enclaves with TCS [14].

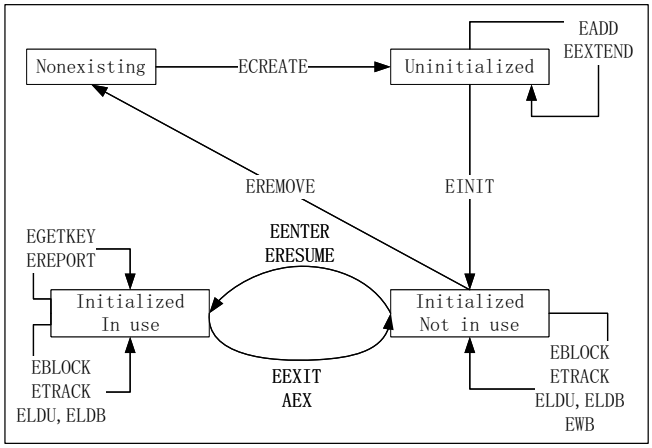

Fig. 1. The Disgram of Intel SGX Enclave Life Cycle Managment Instructions and State Transition

Each enclave is associated with at least one TCS structure that indicates the point of execution into the enclave. Since Intel SGX supports multi-threading, enclave can have as many active threads as TCSNum. Each enclave instance can have many Worker Threads that specifically execute enclave functions. Each thread corresponds to a TCS, which is used to describe the information of this thread [15].

C. About Thread Control Structure (TCS)

Intel SGX stores the current context in the enclave in the TCS, and stores related information in the State Save Area (SSA). TCS and SSA exist in a portion of EPC memory, which is encrypted memory that only the exact instance of the enclave can access. When an enclave is first loaded, the Intel SGX platform reserves encrypted memory for the enclave. This encrypted memory including any space required by the TCS used by the enclave. This forms part of the enclave memory layout,which is fixed and signed during the enclave build, so once deployed cannot be modified. The Architecture field in the TCS lists the context switches that the logical processor performs when transitioning between executing non-enclave code and enclave code. Each logical processor that executes the enclave code uses a TCS. When a TCS is used by a logical processor, it is marked as busy and it cannot be used by any other logical processors.

EENTER takes the virtual address of the TCS as input, requires that the TCS is available (not busy), and at least one SSA in the TCS is available. The check available for SSA is accomplished by ensuring that the Current SSA (CSSA) index field of the TCS is less than the value of the Number of SSA (NSSA) field. In the case of a hardware exception when executing the enclave code, the SSA specified by CSSA will be used.

D. Enclave multi-threaded concurrency security risks

The TCS uses a field to mark whether it enters the busy state, which ensures that the logical CPUs enter and use the same TCS synchronously. Since at least one available SSA is required in the TCS before the logical CPU enters the TCS

to perform the enclave function, the actual number of TCSs is limited by the NSSA. Both TCS and SSA are stored in the EPC. The size of the Processor Reserved Memory (PRM) to store EPC is limited, resulting in a limited size of NSSA. The size of NSSA is limited by Limitation results in a limited number of TCS. In addition, each TCS in the busy state will also occupy a logical CPU. The number of logical CPUs is equal to the number of physical CPUs multiplied by the number of cores (overclocking state multiplied by 2). It can be seen that the number of logical CPUs also exists upper limit. Based on the above analysis, this paper proposes the possible security risks of logical CPU scheduling and TCS use in the case of multiple concurrency.

- In the case of multi-threaded concurrency, whether there will be unreasonable scheduling of the logical CPUs that cause some threads to fail to execute.
- In the case of multi-threaded concurrency, when the size of TCSNum set by the enclave author remains unchanged and a single enclave is subject to flooding attacks, the task thread of the enclave author may execute abnormally.
- In the case of multi-threaded concurrency, when a single enclave is flooded, it may affect the correct execution of other enclaves that are not flooded.
- In the case of multi-threaded concurrency, when the number of enclave instances reaches the upper limit due to the number of EPCs, whether the thread that fails to create an enclave instance will affect the concurrent execution of multiple enclave instances that have been created successfully.

IV. Analysis of multi-threaded concurrency security of enclave in Intel SGX

A. Logical CPU Scheduling under Enclave Multi-Thread Concurrency

The threads running in the enclave use the same threads as the user application. The operating system is responsible for all thread scheduling. The main difference is that when a thread runs inside enclave, it uses the internal stack and thread descriptor of enclave, which is different from the stack and thread descriptor allocated by the operating system when the thread runs outside [16]. An application can create more threads than the number of logical processors, but can only run threads with the same number of logical processors in parallel at a time. The operating system schedules all threads according to the relevant scheduling policies to ensure that all threads are executed as much as possible. Without knowing what specific scheduling strategy the operating system adopts for this type of thread (enclave thread). Whether there is a security risk in the scheduling strategy adopted for the logical CPU will be verified by designing experiments in the following chapters.

B. Enclave suffers flood attack

The number of concurrently running threads in an enclave is limited by the number of TCS. TCS is limited by the amount of EPC memory available. Since at least one SSA is required in the TCS before the logical CPU enters the TCS to execute the enclave function. The number of TCS is also limited by the NSSA. The memory model of Intel SGX technology is shown in Fig 2 below.

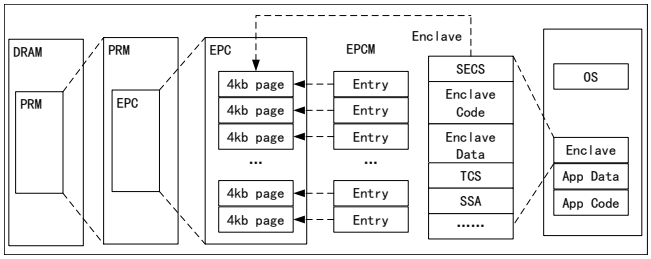

Fig. 2. Memory Model Diagram for Intel SGX Technology

Theoretically, since there is an upper limit on the size of tcsnum, the attacker knows the enclaveid by default. When the author of the enclave sets the value of tcsnum , as long as the attacker executes enough concurrent threads, the related resources of enclave will be completely consumed, and some threads will not execute correctly. Therefore, in theory, in the case of flooding attacks, the enclave does not guarantee that the thread that the author of the enclave wants to execute will be executed. In the case of multi-threaded concurrency, when the size of tcsnum set by the author of the enclave is determined, and the enclave encounters a flood attack. Whether the thread that the author wants to execute can execute correctly will be designed and verified in the following chapters.

C. Logical CPU Scheduling and Flood Attacks under Enclave Multi-Thread Concurrency

Based on the analysis of the above sections A and B. In the case of multi-threaded concurrency, when a single enclave is flooded, will it affect the correct execution of multiple other enclaves that have not been flooded. In theory, the enclave creation action is just to convert a free EPC page to the SECS of the new enclave and mark the enclave as uninitialized in the SECS. When the creation of multiple enclaves is successfully completed and multi-threaded concurrent execution is performed, multiple idle logical processors will execute the EADD and EENTER instructions in parallel to complete the execution of the enclave. The unexecuted enclave will continue to execute EADD and EENTER after the logical CPU is released. If an enclave encounters a flood attack at this time, other enclaves will continue to complete the EADD and EENTER of other enclaves after the logical CPU is released to complete the execution. In the case of multi-threaded concurrency, when a single enclave encounters a flooding attack, whether it will affect the correct execution of other enclaves that have not suffered a flooding attack will be verified by designing experiments in the following chapters.

D. Restrictions on the creation and execution of enclave under multi-threaded concurrency

The successful creation of an enclave is a prerequisite for the execution of an enclave. An enclave that has been created successfully can be executed correctly, while an enclave that fails to be created will not be executed without the basis for execution. In theory, an enclave thread that fails to create

will throw an exception and terminate the thread, and will not affect the execution of multiple enclave instances that are successfully created. In the case of multi-threaded concurrency, when the number of enclave instances reaches the upper limit due to the limit of the number of EPC. Whether the thread that fails to create an enclave instance will affect the concurrent execution of multiple threads that successfully create an enclave instance will be verified in the following chapters by designing experiments.

V. Experimental Results and Analysis

A. Development environment

The experimental environment in this paper is showed as follow table 1.

TABLE I
Experimental Environment Configuration

Configuration name	Detailed parameters
Processor	Intel(R)Core(TM)i5-6300HQ CPU @2.30GHz
Operating system	Windows10 64-bit
Memory	16.0GB
Intel SGX SDK	Intel SGX SDK for Windows v2.14.101.1
IDE	Visual Studio 2019
Development Language	c/c++ language

B. Experiment of logical CPU scheduling security under enclave multithreading

Aiming at the question of whether the scheduling strategy adopted by the operating system for the logical CPU in the case of enclave multi-threaded concurrency is a security risk. The following experiments are designed.

1) Experiment One: Set TCSNum=1. Mark the number of logical CPU as m. Mark the upper limit of the number of enclave instances that can be created by the current hardware environment as s. Create n enclave instances($m \ll n \leq s, n \in z$). Create n one-to-one corresponding concurrent threads for the successfully created n enclave instances. Each thread executes the same enclave entry function. One of the representative running results is as Fig 3.

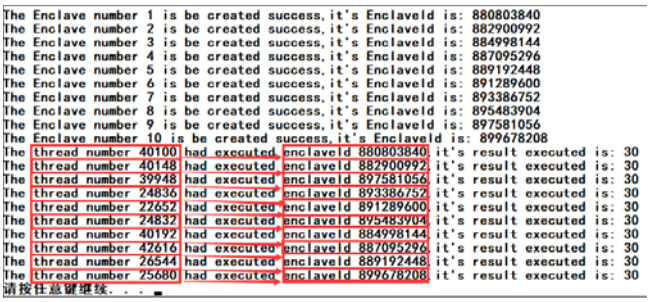

Fig. 3. Experiment One result chart

Graphical description of the results of Experiment One as shown in Fig 4.

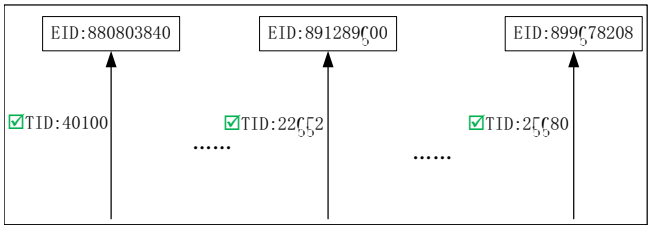

Fig. 4. Graphical description of experimental results

In this experiment, the value of n is continuously adjusted, and the conclusion is drawn after running multiple runs and analyzing the running results. It is not found that the scheduling strategy adopted by the operating system for logical CPUs have security risks in the case of multi-threaded concurrency in enclave.

C. Enclave suffered flood attack experiment

In the case of multi-thread concurrency, when the size of TCSNum set by the author of the enclave, and the enclave is attacked by flooding, whether the thread that the author of the enclave hopes to execute can be executed correctly. The following experiment is designed.

Experiment Two: Set TCSNum=2. Create an enclave instance. Create n concurrent threads ($n \geq 2, n \in z$) for the successfully created enclave instance. Each thread executes the same enclave entry function. one of which the representative running results are as follow Fig 5 (n=20). Graphical description of the experimental results is showed as Fig 6.

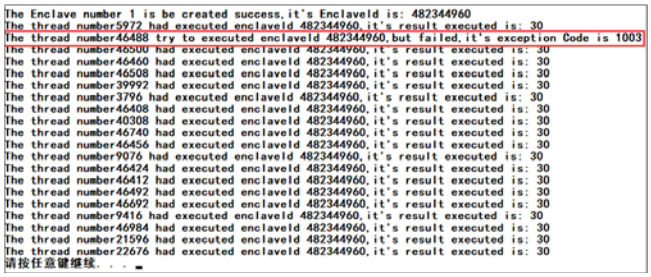

Fig. 5. The result of experiment two

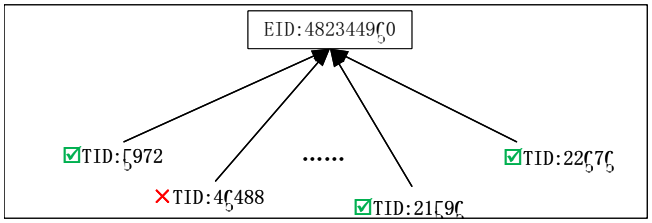

Fig. 6. Graphical description of the results of experiment two

In this experiment, the value of n is adjusted continuously, and the conclusion is drawn after many experiments and analysis of the experimental results.

When TCSNum is set to 2, in the case of multi-threaded concurrency, there is no guarantee that the thread that the enclave author wants to execute will be executed correctly. The thread in which the error occurred has an error code of 0x1003.

When the number of concurrent threads is 200000, the graph of TCSNum value and abnormal thread (0x1003) is shown in Fig 7.

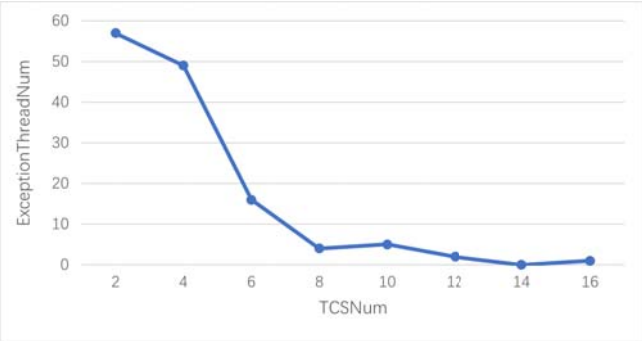

Fig. 7. Relationship between TCSNum and abnormal thread

In the current development environment, after adjusting the value of n in experiment two several times. Tt is found that when $TCSNum \geq 12$, the enclave can run smoothly under 200,000 concurrency. Which almost guarantees that the thread that the author of the enclave wants to execute is executed correctly. Different development environments may have different critical values. When the size of tcsnum is fixed, what is the relationship between the concurrency and the number of abnormal threads. The specific experiments are as follows.

When TCSNum=6, the relationship between concurrency and ExceptionThreadNum is shown in Fig 8.

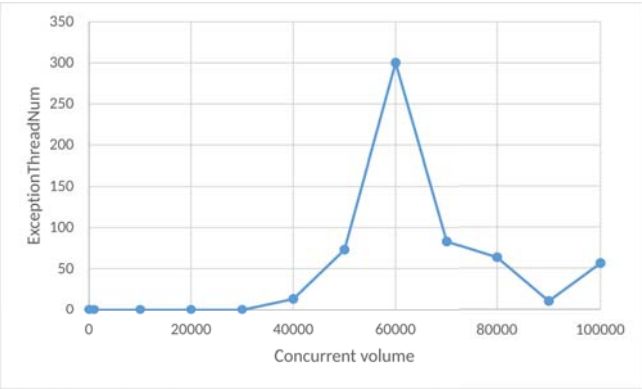

Fig. 8. The result of experiment three

It can be seen from Fig. 8 that when the size of tcsnum is fixed, the number of abnormal threads will reach the peak at a specific concurrency. From the above related theoretical analysis, it can be seen that under a certain PRM size, the larger the tcsnum value, the smaller the upper limit of the enclave instance that can be created. Under the conditions determined by PRM, that is, the development environment remains unchanged, find the balance between the tcsnum value and the upper limit of the number of enclaves. The balance point shall meet the following conditions at the same time.

1) Meet the requirements of the enclave author for the maximum number of concurrently active enclaves.
2) Maximize the TCSNum value, thereby minimizing the probability of a single enclave instance suffering losses caused by flooding attacks.

In the current environment, by continuously increasing the value of TCSNum and repeating the experiment, the relationship between TCSNum and MEEN is obtained. The specific relationship is shown in the following figure 9.

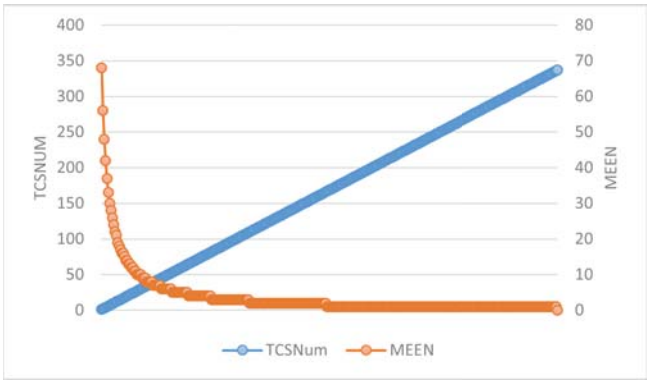

Fig. 9. Relationship between TCSNum and MEEN

Through a comprehensive analysis of the above experiments, when an enclave encounters a flood attack, the value of TCSNum can be adjusted appropriately to find a balance between the TCSNum value and the upper limit of the number of enclaves. So as to minimize the probability of losses caused by flood attacks but cannot be completely avoided. That is, when an enclave encounters a flood attack, there is no complete guarantee that the thread that the author of the enclave wants to execute will be executed correctly.

D. Cross-experiment of logical CPU scheduling and flooding attack under enclave concurrency

In the case of multi-threaded concurrency, when a single enclave encounters a flooding attack, will it affect the correct execution of other enclaves that have not suffered a flooding attack, the following experiments are designed.

Experiment Three: Set TCSNum=2. Mark the number of logical CPUs as m. Mark the upper limit of the number of enclave instances that can be created by the current hardware environment as s. Create n enclave instances ($m \ll n \leq s, n \in z$). Randomly select a successfully created enclave instance, and create t($t \gg TCSNum$) threads for this instance. Create (n-1) one-to-one corresponding concurrent threads for the remaining (n-1) successfully created enclave instances. Each thread executes the same enclave entry function. In the experiment, the values of TCSNum, n and t are continuously

adjusted. After repeated operation for many times, the graphical description of the experimental results is as Fig 10.

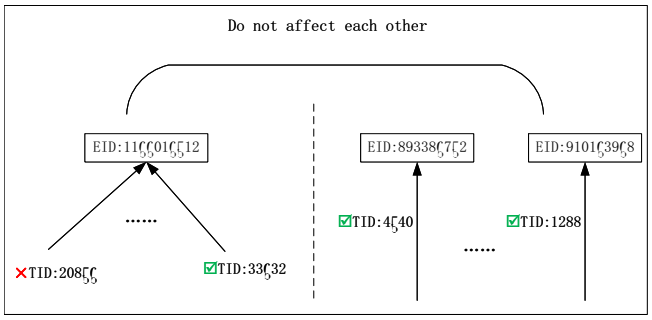

Fig. 10. Experiment three Graphical Description

According to the experimental results, the following conclusions can be drawn:

In the case of multi-threaded concurrency, when a single enclave is flooded, it will not affect the correct execution of other enclaves that are not flooded.

E. Experiment on the creation and execution constraints of enclave under multi-threaded concurrency

In the case of multi-threaded concurrency, when the number of enclave instances reaches the upper limit due to the limit of the number of EPC. Whether the thread that fails to create an enclave instance will affect the concurrent execution of multiple threads that successfully create enclave instances. The following experiments are designed.

Experiment Four: Set TCSNum=1. Record the upper limit of the number of enclave instances that can be created by the current hardware environment as s. Try to create n enclave instances ($n > s, n \in z$). Create n one-to-one corresponding concurrent threads. Each thread executes the same enclave entry function. One of the representative run results is as Fig 11(n=69).

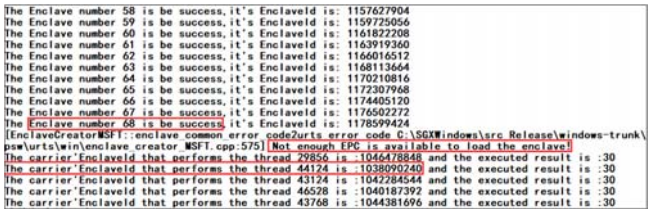

Fig. 11. The result of experiment six

According to repeated experiments, it can be concluded that in the case of multi-threaded concurrency, when the number of enclave instances reaches the upper limit due to the limit of the number of EPC. The thread that fails to create an enclave instance will not affect the concurrent execution of multiple threads that successfully create an enclave instance.

VI. Conclusion

This article focuses on the security risks that may exist in the creation and use of enclaves in the case of multi-threaded concurrency. There are four specific problems in total. This paper analyzes them and designs experiments to verify the final conclusion. It is found that in the case of multi-threaded concurrency, a single enclave cannot resist flooding attacks, and related threads also throw TCS exception codes.

Acknowledgments

This research work is supported by the National Natural Science Founds of China (62072368, U20B2050), Key Research and Development Program of Shaanxi Province (2021ZDLGY05-09, 2022GY-040).

References

[1] Costan, Victor, and Srinivas Devadas. "Intel SGX explained." Cryptology ePrint Archive (2016).
[2] Wang Juan, et al. "Analysis and Research of SGX Technology." Journal of Software 29.9 (2018): 2778-2798.
[3] Cloosters, Tobias, Michael Rodler, and Lucas Davi. "TeeRex: Discovery and Exploitation of Memory Corruption Vulnerabilities in SGX Enclaves." 29th USENIX Security Symposium (USENIX Security 20). 2020.
[4] W. Wang, W. Liu, H. Chen, X. Wang, H. Tian and D. Lin, "Trust Beyond Border: Lightweight, Verifiable User Isolation for Protecting In-Enclave Services," in IEEE Transactions on Dependable and Secure Computing, doi: 10.1109/TDSC.2021.3138427.
[5] B. D'Agostino and O. Khan, "Seeds of SEED: Characterizing Enclave-level Parallelism in Secure Multicore Processors," 2021 International Symposium on Secure and Private Execution Environment Design (SEED), 2021, pp. 203-209, doi: 10.1109/SEED51797.2021.00031.
[6] Shen, Youren, et al. "Occlum: Secure and efficient multitasking inside a single enclave of intel sgx." Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 2020.
[7] Weichbrodt, Nico, et al. "AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves." European Symposium on Research in Computer Security. Springer, Cham, 2016.
[8] Gu, Jin-Yu, Hao Li, and Zheng-Yu He. "Unified Enclave Abstraction and Secure Enclave Migration on Heterogeneous Security Architectures." JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37.2 (2022): 468-486.
[9] T. Yavuz, F. Fowze, G. Hernandez, K. Y. Bai, K. Butler and D. J. Tian, "ENCIDER: Detecting Timing and Cache Side Channels in SGX Enclaves and Cryptographic APIs," in IEEE Transactions on Dependable and Secure Computing, doi: 10.1109/TDSC.2022.3160346.
[10] Kockan, Can, et al. "Sketching algorithms for genomic data analysis and querying in a secure enclave." Nature methods 17.3 (2020): 295-301.
[11] E. Antolak and A. Pułka, "Energy-Efficient Task Scheduling in Design of Multithread Time Predictable Real-Time Systems," in IEEE Access, vol. 9, pp. 121111-121127, 2021, doi: 10.1109/ACCESS.2021.3108912.
[12] Hei, Xinhong, et al. "From hardware to operating system: a static measurement method of android system based on TrustZone." Wireless Communications and Mobile Computing 2020 (2020).
[13] Y. Wang, W. Gao, X. Hei, I. Mungwarama and J. Ren, "Independent credible: Secure communication architecture of Android devices based on TrustZone," 2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), 2020, pp. 85-92, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics50389.2020.00032.
[14] The SGX design fully supports multi-core processors, and each logical processor can execute the code of the same enclave through different threads.
[15] Shinde, Shweta, et al. "Binary Compatibility For SGX Enclaves." arXiv preprint arXiv:2009.01144 (2020).
[16] Choi, Changho, et al. "S-OpenSGX: A system-level platform for exploring SGX enclave-based computing." computers and security 70 (2017): 290-306.