Forking Attacks: Two Ways, No Worries

Matija Roncevic
matija.roncevic@fau.de
Friedricht-Alexander-Universität Erlangen-Nürnberg

1 Abstract

A summary of the paper

2 Introduction

Almost every noteworthy application has embraced cloud computing. The range of applications that benefit from cloud services spans from small smartphone apps to sophisticated large language models to commonly used GitHub repositories. The clear reason for its popularity is the ability to store, manage, and process data more efficiently than ever before. The shift to cloud services offers numerous advantages, including scalability, flexibility, and cost-effectiveness. However, it also introduces significant security challenges. With the inclusion of numerous critical applications and sensitive data, the importance of secure environments has increased. Cloud security involves a diverse range of practices, technologies, and policies aimed at protecting data, applications, and services distributed across cloud environments. The baseline of protection is realized by implementing firewalls and encryption; however, the sophisticated and evolving nature of cyber threats often outpaces the safety measures these solutions provide. This calls for advanced security solutions that can safeguard data even in potentially compromised environments.

Trusted Execution Environments (TEE) like Intel Software Guard Extensions (SGX) is such an advanced security solution. Intel SGX is a set of hardware-based security features that create isolated execution environments, known as enclaves [1].

(Enclaves are secure areas of memory where sensitive data and code can be executed and stored in isolation from the rest of the system.)

These enclaves are designed to protect sensitive data and code from being accessed or modified by unauthorized parties, even if the operating system is compromised. SGX provides a robust mechanism for ensuring the confidentiality and integrity of data processed in the cloud. Despite its strengths, Intel SGX still has its vulnerabilities. The enclaves themselves are in fact isolated from adversaries, however, the reliability of the inputs can not be ensured without additional safety measures. Those attacks are labeled as rollback attacks [2]. (More explanation about rollbacks, maybe in the background?)

Forking attacks, on the other hand, aim to create multiple instances of an enclave, leading to unauthorized data access and manipulation. This is accomplished by running those simultaneously and exploiting the fact, that all those enclaves will return correct but often stale states. I.e. counters for password attempts can be reset to gain unlimited tries, despite the limit for tries being set to a finite number. Understanding these attacks and developing effective mitigation strategies is essential for ensuring the security and trustworthiness of cloud-based applications relying on SGX.

————————————- Grammarly checked ————————————

3 Background

The concepts that are discussed in this paper will be explained further to provide comprehensive knowledge beforehand.

3.1 Trusted Execution Environment, TEE

TEEs aim to increase the safeness of data and exacerbate the tempering with crucial code sections, by allowing the processor to have protection through different implementations like Intel SGX enclave system.

3.2 Attacks

3.2.1 Denial of Service (DoS) Attacks

3.2.2 Forking Attacks

3.2.3 Rollback Attacks

————————————- Grammarly checked ————————————

4 Body

This section will introduce two ways to mitigate forking attacks. Each method has its implementations for tackling the challenges, thereby bringing advantages, disadvantages, and preferable fields of application.

4.1 The Blockchain approach - Narrator-Pro

This system relies on an external blockchain to initialize certain components. The system itself can be broken down into three main concepts which combined yield in its prevention of attacks. The goals that are ensured are described as:

- Security - The safety and liveness properties of the TEE programs will be protected/guaranteed.
- Performance - While providing the Security Goals there will be no decisive detriment to performance. In detail low latency for state updates and read operations, high throughput for processing enclave program requests and unlimited state updates, provided by the blockchain. //Check thruthness

Before getting into the concepts involved, it is essential to give an overview of the system so the coherences will be clear.

Several SGX-enabled machines are running in a cloud. These machines can run a number (limited by specifications of the system) of enclaves which are divided into two groups of Application Enclaves (AEs) and State Enclaves (SEs). AEs have applications running, handle client requests, and return outputs corresponding to their inputs. SEs on the other hand contain the Narrator-Pro software and are responsible for providing state continuity to AEs. This is accomplished by a secure connection from the AE to a locally running SE, where the AE can use SEs Narrator-Pro libraries to seal data. This data is then used to retrieve the latest sealed state, in case of unexpected shutdowns. This principle is explained in more detail

in section .... -Figure 1- provides an overview of the mentioned components.
The creators also state a few ...(premises) which Narrator-Pro does.

- **Denial of Service Attacks** - It is not the goal to prevent systems from these kinds of attacks. Since TEEs themselves do not have prevention measures included.
- **Hardware** - The implementation of Narrator-Pro should neither require any hardware changes nor will there be a need for specific hardware if the cloud TEE is already running.
- **No Trusted Central Party** - ...
-

The main concepts which constitute the reliance of Narrator-Pro are (1) system initialization §..., (2) state update and read protocols for AEs §... and (3) restart protocols regarding AEs and SEs §....

4.1.1 System Initialization

Before the system can proceed with its actual work //different Word//, it is essential to initialize the SEs. Any adversaries will not be able to launch SEs with the same binary on different channels to get stale states. This is accomplished by the usage of a blockchain *B*.
The blockchain *B* serves to store key-value tuples containing <*key*, *value*>. The *value* is a random string linked to the *key*, which will be returned by the blockchain in future read-calls. //However to verify the recieved *value*, *B* proviedes an authenticator *a* in the initialization process.The second section, where an authenticator *a* is returned, is while initialization//Important??// However, the, before mentioned, initialization process of SEs is carried out while the first wirte-call from a SE to *B* is established. *B* will verify that no other SE has registered on the blockchain with the provided *key*. If this is the case the write-call will be executed and the client will receive an authenticator *a*, with which he can verify the operation. This summarises the initialization responsibilities of *B*.
The SE on the other hand, has to go through additional steps to finish its initialization. Therefore it is important to mention that SEs connect in groups to carry out requests delivered by AEs. Every group has an SE leader who is responsible for storing certain information about the group. So if an SE wants to initialize it first lets the leader establish a secure connection to itself. This connection channel spreads through the whole group so the participants can talk to each other. After the channel is in place, the initializing SE creates a key-pair (which will be used in future connection establishments with this SE), sends it to the leader and it will be collected and stored in a list that is accessible for every SE in the group. After that, the before mentioned blockchain interaction takes place where the SE performs a read-call to *B* and either receives the *value* containing *Null* (stating that no other SE has already linked to it with this *key*) and the initialization succeeds or the SE terminates its initialization process. Subsequently, the first write-call will be executed where the SE stores its <*key*, *value*> tuple, and only then it can proceed to process the AE request.

4.1.2 State Update and Read

To start this section it's essential to mention that Narrator-Pro considers a distributed system if it at least holds $n = 2f + 1$ SGX-enabled machines in a cloud. Where f is the number of faulty SEs, so the system can still properly work.
To prevent eavs dropping from adversaries, messages between AEs and SEs are transmitted over encrypted channels, using secret session keys. A nonce (Number used once) is sent as well to prevent replays. An AE will use an aforementioned group of SEs to confirm its current state S_i, before proceeding with input I_i to update to S_{i+1}. The following paragraphs will explain this procedure further (SD_i is called the state digest which is a summary of the latest states):

(1) **Input** - After receiving I_i from a user, the AE saves a state snapshot <S_i, I_i, SD_{i-1}> on the disk, in case the process crashes midway, so the current input/state can be retrieved. The AEs next step is to call the function (**writeState(**SD_i**)** -> ACK) and wait for the returning ACK from the SE, to proceed with the state update.
(2) **writeState()** - With calling the **writeState()** function, the SE saves a snapshot of its states, containing <S_j, I_j, SD_{j-1}> with $I_j = SD_i$. Upon a successful save, the SE starts communicating with its group by sending a PREPARE message <Prepare, SD_j, (*j*, *seq*)> to all SEs.
(3) **PREPARE message** - Receiving a PREPARE message triggers the SE to update the sending SE state digest in the memory. Consequently, an ECHO message is sent, containing SD_j, signaling a successful update to the sending SE. Upon receiving f + 1 ECHO messages, the initial SE adds the new state S_{i+1} to I_{j+1} and sends an ACK to the AE, which triggers the evaluation to S_{i+1} and the corresponding output to the user.

A similar approach is used when an AE wants to verify the freshness of its current state. It calls the **readState()** function of its connected SE. The target SE, however, can not just return its current state digest (containing the latest states, with the freshest on top) since due to a forking attack there could be more than one instance of this SE, holding stale state digests. So the creators of Narrator-Pro developed a sequence this SE has to run through, to verify its freshness:

(1) After calling **readState()** the target SE sends a request to all SEs in the group to obtain their saved state digests.
(2) Each SE then sends back the saved state digest, corresponding to the target SE, since every SE holds a list of all SE state digests, in the group.
(3) The target SE has to receive at least f + 1 replies, where the state digest is the same as its, to be able to return it to the AE. Otherwise, it has to collect the sealed data from the OS to resume its latest state. //Meaning??//

4.1.3 Restart Protocol

// give a simple summary of the Restart Protocol?
// To end this section, give a small summary about the counter measures/last 3 sections, Narrator-Pro delivers?

4.2 Cache-based Channel - CloneBusters

The uniqueness of CloneBusters is the resignation of reliance on a Trusted Third Party since they are hard to find in real-life applications. To achieve this goal they implemented a channel (respectfully a cache set) for every binary, i.e. enclaves (they have certain binaries designated, if an enclave is forked the binary stays the same therefore the same channel) in which only his binary can communicate. This communication is then monitored by CloneBusters and

fed to a classification algorithm to detect clones of an instance i.e. enclave. The creators of CloneBusters also make some assumptions beforehand to set an environment in which it will perform in the expected way:

- **Clones** - If a second instance of an enclave is generated it is set to be a clone if (i) they are loaded with the same binary and (ii) they run at the same time. (ii) is necessary since SGX itself can detect the clone if the clones run one at a time. Here the system relies on Monotonic Counters (MC) which will be incremented by an enclave to track its progress. The mode *"inc-then-store"* is used which can be described as when data is stored the index will be increased beforehand and saved as well to deliver validation. This validation is carried out once while storing the data so no index can be assigned to multiple data sets and second, if an enclave requests a data set it only accepts it if its counter is the same as the counter saved along with the data.
- ...

There was a vulnerability left out in the above definition of clones and the explanation of the *"inc-then-store"* mode. Immagin one enclave E, its clone E', and a malicious OS that created that clone and is in control of the execution order of E and E'. Both enclaves will start with the same MC value (which is global) and loaded binary. In order to process Input I_1, E increments MC once, however is then paused through the OS. The OS then feeds I_2 to E' which also increments MC, leading MC to be +2. After that, the enclaves can proceed simultaneously since they will execute **Read(MC)** and receive the same value but arrive at different outputs which will be stored with the same index leading to a violation of the given rollback attack prevention from SGX. Therefore CloneBusters developed a strategy to detect clones as they try to operate parallel to its enclave. The implementation of communication through already available cache-sets enables CloneBusters to not rely on a TTP. This is accomplished by two phases which have to be executed, (1) the preparation phase and (2) the monitoring phase. The group of cache sets, i.e. the channel, is developed in (1) to enable phase (2) to monitor this channel where every enclave with the same binary (the enclave and its clones) will put its data in so a classification algorithm can check cache hits and misses to detect clones which will inevitably oust the data from the original enclave with its signed data.

————————————- Grammarly checked ————————————

4.2.1 Preparation Phase (1)

In order to assign cache sets to enclaves the creators of CloneBusters evalutated the physical addresses (16 Bits) of the system and came to an conclusion that these sets are best to be indexed by bits 6-15. This leaves a total of 16 cache sets which are used for a channel of an enclave. In the paper they show that this is the perfect size for encalves to detect the presens of clones, whitout providing the OS a chance to temper with that //different wording//. After determining the rigth quantity, eviction sets have to be allocated in order to finalize the channels in which the enclaves will communicate in. The encalves should have eviction sets where the memory is contiguous,howevery the OS has the power to assign non-contiguous memory. In order avoid this happening //different wording//, the creators developed an algorithm to create the eviction sets properly. The algorithm and its method of operation can be detailed in the paper.

4.2.2 Monitoring Phase (2)

With the establishment of the eviction sets the enclaves can start operating. CloneBusters has to run continuously while a critical phase is executed(e.g. while incrementing and reading the MC), to provide optimal security. Before entering the *"inc-then-store"* section, the enclave has to pre-fetch, the data to be monitored, into the cache set to provide expressive reliance of CloneBusters. The monitoring is done by measuring the time to access data on the cache. This data is then forwarded to a classification algorithm. With a pre-defined threshold of what time it takes to access eather cache hits or cache misses, the algorithm can distinguish hits and misses and therby determine of a clone is running. This information is the passed back to the enclave to take appropriate counter measures like holding the execution of the enclave.

5 Evaluation

Evaluating the secutry analysis from both papers. Point out the most important parts?

6 Conclusion

References

[1] Samira Briongos, Ghassan Karame, Claudio Soriente, and Annika Wilde. No forking way: Detecting cloning attacks on intel sgx applications. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23, New York, NY, USA, 2023. Association for Computing Machinery.

[2] Wei Peng, Xiang Li, Jianyu Niu, Xiaokuan Zhang, and Yinqian Zhang. Ensuring state continuity for confidential computing: A blockchain-based approach. *IEEE Transactions on Dependable and Secure Computing*, pages 1–14, 2024.