

Forking Attacks: Two Ways, No Worries

Matija Roncevic

matija.roncevic@fau.de

Friedrich-Alexander-Universität Erlangen-Nürnberg

1 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 own vulnerabilities. The enclaves itself are in fact isolated from adversaries, however the reliability on the inputs can not be ensured without additional safety measures. Those attacks are labeled as rollback attacks [2]. (More explanation about rollbacks, maybe in background?)

Forking attacks on the other hand, aim on creating 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 is set to an 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.

2 BACKGROUND

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

2.1 Trusted Execution Environment, TEE

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

2.2 Denial of Service (DoS) Attacks

2.3 Liveness

2.4 Safety

3 BODY

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

3.1 The Blockchain approach - Narrator-Pro

This system relies on an external blockchain to initialize certain components. The system itself can be broken down to three main concepts which combined yield in its prevention of attacks. The goals which 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 of 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 thoroughness

Before getting into the concepts involved, it is essential to give an overview on 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 its inputs. SEs on the other hand contain the Narrator-Pro software and are responsible to provide 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 shut downs. 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 preventing measures included.

- **Hardware** - The implementation of Narrator-Pro should neither require any hardware changes nor will there be a need of specific hardware if the cloud TEE is already running.
- **No Trusted Central Party** - ...
-

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

3.1.1 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 $\langle key, value \rangle$. 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 received *value*, B provides an authenticator a in the initialization process. The second operation, where an authenticator a is returned, is while initialization //Important???// However, the before mentioned, initialization process of SEs is carried out while the first write-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 summarizes 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 which is responsible to store certain information about the group. So if a SE wants to initialize it firstly lets the leader establish a secure connection to it self. 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 which 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 $\langle key, value \rangle$ tuple and only then it can proceed processing the AEs request.

3.1.2 State Update and Read. To start this section its essential to mention that Narrator-Pro considers a distributed system if it atleast 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 eavesdropping 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 $\langle S_i, I_i, SD_{i-1} \rangle$ on the disk, in case the process crashes midway, so the current input/state can be retrieved. The AE's next step is to call the function (**writeState**(SD_i) \rightarrow 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 own states, containing $\langle S_j, I_j, SD_{j-1} \rangle$ with $I_j = SD_i$. Upon a successful save, the SE starts communicating with its group by sending a PREPARE message $\langle \text{Prepare}, SD_j, (j, seq) \rangle$ to all SEs.
- (3) **PREPARE message** - Receiving a PREPARE message triggers the SE to update the sending SEs 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 evolution to S_{i+1} and the corresponding output to the user.

An 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 SEs 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???//

3.1.3 Restart Protocol. // give a simple summary of the Restart Protocol?

// TO end this section, give a small summary about the counter measures Narrator-Pro delivers?

3.2 Cache-based channel against Forking Attacks - CloneBusters

4 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.