Forking Attacks: Two Ways, No Worries

Matija Roncevic
matija.roncevic@fau.de
Friedricht-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 [**?**].
(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 relaiability on the inputs can not be ensured without additional savety meassures. Those attacks are labeled as rollback attacks [**?**]. (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 simulainously and exploiting the fact, that all those enclaves will return correct but often stale states. I.e. counters for passwort 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 provied an comprenhensive knowledge beforhand.

2.1 Trusted Execution Environment, TEE

TEEs aim to increae the saveness of data and exacerbate the tempering with crucial code sections, by allowing the processor to have protection through different implementations like Intels 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 programms will be protected/guranteed.
- 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 provessing enclae program requests and unlimited state updates, provided by the blockchain. //Check thruthness

Before getting into the concepts involed, it is essential to give an overview on the system so the coherences will be clear.
Several SGX enabled maschines are running in a cloud. These maschines can run a number (limited by specifications of the system) of enclaves which are devided 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 lokaly 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 priciple is expained 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 includet.

- **Hardware** - The implementation of Narrator-Pro should neader 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. Befor 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 an 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 ecation, 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 recieve an authenticator *a*, whit which he can verify the operation. This summerices the initialization responsiblities 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 an secure connection to it self. This connection channel spreads thorugh the whole group so the participents can talk to each other. After the channel is in place, the initilazing SE creates an 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 accesseble 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 eather recieves the *value* conatining *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 processing the AEs request.

3.1.2 State Update and Read. To prevent eavs droping from adversaries, messages between AEs and SEs are transmitted over encrypted channels, using secret session keys. 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}. A nonce (Number used once) is send as well to prevent replays. The following paragraphs will explain this procedur further (SD_i is a summary of the latest states):

(1) After recieving I_i from an user, the AE saves an 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 funktion (**writeState(**SD_i**)** -> ACK) and wait for the returning ACK from the SE, to proceed with the state update.
(2) With calling the **writeState()** function, the SE saves an snapshot of its own states, containing <S_j, I_j, SD_j> with I_j = SD_i. Upon a successfull save, the SE starts communicating with its group by

4 CONCLUSION