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. 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. Basic protection comes from using firewalls and encryption, but cyber threats are getting so advanced and constantly changing that these measures can't always keep up. This calls for advanced security solutions that can ensure the integrity of data even in potentially compromised environments.

In order to protect critical sections of programs, Trusted Execution Environments (TEE) were developed. These TEEs have safe areas of memory where critical sections of programs can be executed in isolation from the rest of the system i.e., Enclaves. Software Guard Extension (SGX), invented by Intel, falls into the category of a TEE and has Enclaves implemented [2]. These enclaves 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 input can not be ensured without additional safety measures. So a malicious OS could feed the enclave stale input data, to restore a passed state. Those attacks are labeled as rollback attacks, where data is rolled to an older version [4].
Forking attacks, on the other hand, aim to create multiple instances of an enclave (Clone), leading to unauthorized data access and manipulation. This is accomplished by running the enclave and the clones simultaneously and exploiting the fact, that all those instances will return correct but stale states. E.g., 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. [1].
In this paper, we will analyze two methods that provide countermeasures against forking attacks. The authors of the first system, Narrator-Pro, relied on a blockchain initialization and developed additional processes for enclaves. The second paper, CloneBusters, on the other hand, created cache-based channels in which enclaves have to communicate, in order to detect any clones.
———————————- Grammarly checked ———————————

3 Background

The following section will provide further knowledge about different aspects, covered in this paper.

3.1 Trusted Execution Environment, TEE

TEEs are designed to increace data security and prevent tampering with critical code sections. They achieve this by applying various protective mechanisms, such as the Intel SGX enclave system. The processor is, in those systems, provided with several memory areas which are ensured to be separate and safe from unauthorized access. This memory space is either detached by hardware or by software provided by Intel SGX. Sections of a program that execute critical operations, such as incrementing essential counters, are executed within these protected environments to ensure their security. Furthermore, adversaries can not read or modify data saved inside this memory since it is encrypted.

3.2 Intel SGX Attacks

This paper aims to provide an overview of two methods to protect systems against forking attacks. Since further attacks were discussed in the papers as well we wanted to give an overview about them. Those also come along when talking about forking and enclaves.

Denial of Service (DoS) Attacks. These attacks differ from the following since the goal here is not to force the system to outputs that deliver advantages to the adversary by feeding in special inputs or launching several instances. DoS attacks rather overflow the system to disable services or downgrade service performance until the system shuts down and, as the name states, deny the completion of certain services [3].

Rollback Attacks. Almost any system with a lifecycle can be rolled back to an outdated state, so an attacker can exploit the output to their advantage. Encalves are not the only targets for those kinds of attacks, VM for example are vulnerable to snapshots. These snapshots can be saved if e.g. certain restrictions on participants are not yet assigned and if needed this snapshot can be executed so this participant can bypass security measures and read information that it should not have access to [5].

Forking Attacks. When an adversary creates several clones of an instance (e.g. an enclave) and launches them into the system, it is labeled as a forking attack. This attack exploits some vulnerabilities that are delivered, e.g. if these clones then try to connect again to the group in which its origin instance is running, it enables the

clone to link to other groups as well, as discussed in §4.1.3. To show the connections between forking and rollback attacks it is important to mention that rollback attacks are often performed on clones of a forking attack. Since the clones deliver the opportunity to feed them stale inputs to receive outputs that bypass different security measures, like limited password attempts. So due to the connection of both attacks, the prevention of forking attacks leads to a certain prevention of rollback attacks as well.

DoS Attacks, on the other hand, are not aimed to be prevented by both methods, as stated later on. The following sections 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.
————————————- Grammarly checked ————————————

4 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 - Achieving the security goals will not significantly impact performance. In detail the aim is low latency for state updates and read operations, high throughput for processing enclave program requests and unlimited state updates, provided by the blockchain.

To start this section it is essential to mention that Narrator-Pro considers a distributed system if it at least holds $n = 2f + 1$ SGX-enabled instances in a cloud. Where f is the number of faulty SEs, so the system can still properly work. The system 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 authors 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** - Narrator-Pro does not rely on Trusted central parties, like trusted servers. However, since an external blockchain is used to initialize the system, so the authors can not state that their system does not rely on Trusted Third Parties.
-

The main concepts which constitute the reliance of Narrator-Pro are (1) system initialization §4.1, (2) state update and read protocols for AEs §4.2 and (3) restart protocols regarding AEs and SEs §4.3.

4.1 System Initialization

Before the system can start processing requests, it is essential to initialize the SEs. 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 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 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. 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.

4.2 State Update and Read

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 refers to 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. The tuple <(j, seq)> will be used in §4.3 for a rebooting SE to indicate if an enclave is active or was displaced by its clone.
(3) **PREPARE message** - Receiving a PREPARE message triggers the SE to update the sending SEs state digest in their 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 SD_{j+1} and sends an ACK to the AE. It then will advance its state to S_{i+1} and sent 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??//

4.3 Restart Protocol

The last vulnerability that was discovered by the creators of Narrator-Pro is while an SE reboots and reconnects to its group, an adversary can leverage the group to perform a certain sequence, to enable forking and/or rollback attacks. This fact is best explained with an example. Consider three SEs S_1, S_2 and S_3. With cloning S_1 -> S_1' and S_2 -> S_2' the initial enclaves will be displaced in the group by its copies, leaving S_1 and S_2 to be inactive. S_3 will now communicate with the clones. However, the enclaves S_1 and S_2 do not know that they are inactive since the initial algorithm does not involve the necessity for the enclaves to know that they were displaced. This enables a copy of S_3 -> S_3' to link to S_1 and S_2. This now leaves two groups, *G1* and *G2* of SEs. An adversary can now increment the MC with *G1* and perform some actions leaving *G2* with the old value of MC and thereby rolling back the application. Or feed to different inputs to S_3 and S_3' and receive different outputs with 2 groups which have, respectfully the same binary, enabling a forking attack. In order to prevent a system from performing this kind of sequence, the creators of Narrator-Pro included an intermediate step where an SE has to be confirmed by its group that at least f SEs are not inactive, when reconnecting. This is accomplished by the tuple <(j, seq)>, which is sent along with the **PREPARE messages**, where the index j refers to the freshness of its state digest and seq is used to distinguish different state digests with the same index.

//summarise the 3 main concepts?//

5 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. Immagine 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.

5.1 Preparation Phase (1)

In order to assign cache sets to enclaves the creators of CloneBusters evaluated the physical addresses (16 Bits) of the system and concluded 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 presence of clones, without providing the OS a chance to temper with that //different wording//. After determining the right quantity, eviction sets have to be allocated to finalize the channels in which the enclaves will communicate in. The enclaves should have eviction sets where the memory is contiguous, however, the OS has the power to assign non-contiguous memory. In order to 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.

5.2 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 on 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 either cache hits or cache misses, the algorithm can distinguish hits and misses and then determine if a clone is running. This information is then passed back to the enclave to take appropriate countermeasures like holding the execution of the enclave.

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

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] Victor Costan and Srinivas Devadas. Intel SGX explained. Cryptology ePrint Archive, Paper 2016/086, 2016. https://eprint.iacr.org/2016/086.

[3] Qijun Gu and Peng Liu. Denial of service attacks. *Handbook of Computer Networks: Distributed Networks, Network Planning, Control, Management, and New Trends and Applications*, 3:454–468, 2007.

[4] 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.

[5] Yubin Xia, Yutao Liu, Haibo Chen, and Binyu Zang. Defending against vm rollback attack. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, pages 1–5, 2012.