Faults In Our Bus: Novel Bus Fault Attacks to Break ARM TrustZone

Anirban Chakraborty, Max Planck Institute for Security and Privacy (MPI-SP), Germany
anirban.chakraborty@mpi-sp.org

***Abstract*—In recent years, Trusted Execution Environments (TEE) have emerged as a promising technology for zero-trust architecture. Due to their strong security guarantees, even in the presence of privileged adversaries, TEEs have become the centerpiece of implementing critical operations in embedded systems. In modern Internet-of-things (IoT) settings, physical attacks such as side-channel and fault attacks are practically relevant due to their nature of deployment and exposure in the wild. Therefore, in order to protect them from physical attacks, a number of techniques such as electromagnetic shields and software checks (memory encryption), are implemented. In this research, however, we demonstrate practical attack scenarios on TEEs using a new dimension: *SoC system bus*. We first unearth the fault characteristics of both aspects of the system bus- data bus and address bus. We then use both data bus and address bus faults to mount an end-to-end attack on a commercial Trusted Execution Environment implementation for embedded systems. Additionally, we also demonstrate loopholes in TEE specification (specifically GlobalPlatform API specification) as well as in Linux function return conventions that aid adversarial objectives. Overall, this research reinforces the importance of considering not only the software implementation of abstract specification but also the execution environment where the implementation is expected to operate.**

I. Introduction

The Internet of Things (IoT) has become an essential part of modern-day life, catering to billions of users across various applications, starting from home appliances to critical services in different industries. Quite naturally, the security aspect of these devices in the IoT hemisphere has intrigued researchers for the past few years. The security model of IoT ecosystem presents newer threats and opportunities as the majority of these embedded devices are deployed *in-the-wild*, mostly without any human supervision, therefore exposing themselves to physical adversaries. In such scenarios, an adversary can mount a plethora of physical attacks, including (but not limited to) fault injection (FI) attacks [**?**]. Fault Injection attacks are a class of active attacks where the adversary, having physical (or remote) access to the system, can perturb intermediate computations of the target algorithm in a seemingly controlled manner. Over the years, the literature on FI attacks has been enriched with myriads of techniques which can be broadly classified into three types - ① manipulating system parameters such as clock frequency or voltage, ② changing external operating environment such as temperature, and ③ injecting external electromagnetic or optical (laser) pulses.

In modern embedded systems market (more specifically in IoT ecosystem) other than FPGAs and ASICs, *Systems-on-Chip* (or SoCs) are being increasingly used. An SoC is a general-purpose computer capable of running a full-fledged operating system (OS) and several applications atop it. Prior works have shown vulnerability of SoCs against faults on processor and memory modules. In response, modern SoCs employ a number of techniques to thwart such attacks. Given the presence of such defences against fault attacks on SoCs, we ask a very fundamental question: *are there other architectural features which can be used for faults, for which no known defences are deployed yet*? In this work, to the best of our knowledge, we provide the first practical demonstration of one such feature: *system bus faults*.

II. Bus Faults: a Novel Fault Attack on SoCs

We introduce *bus faults* - faults injected *while* the system bus is operational, rather than *during* the computation in the processor *after* the operation of the system bus is already over. The system bus has two parts: the *data* bus and the *address* bus. For every memory operation, the memory address to be accessed is placed onto the **address bus**. Likewise, the corresponding data is placed onto the **data bus**. While most of the countermeasures have been focused on either protecting the processor or the memory module, the system bus has not been taken into consideration. As both data (also instruction) and address travel through the system bus, it provides an additional attack surface for FI attacks. We show that a precise fault injection during the execution of `store` and `load` instructions causes corruption of contents of both the address and the data buses. This corruption percolates into execution since the contents of address/data buses drive several operations in the processor and memory. By targeting system bus instead of directly faulting the processor/memory, our bus faults bypass countermeasures preventing processor faults, as well as avoid persistent memory faults (like Rowhammer) that can cause permanent data corruption. In our experiments, we observed that around 30% of times, the data bus fault changes the entire 32-bit register from a non-zero value to `0x0`. We denote this event as *register sweeping* fault model and eventually utilize it to bypass the signature verification check in ARM TrustZone.

III. End to End Attack on ARM TrustZone

A. Attacker Threat Model

Before describing the attacks, we establish goals and assumptions for the adversary. The adversarial objective is two fold: ❶ installing a malicious Trusted Application (TA) by attacking the TEE, and ❷ break symmetrically encrypted

communication channels between CA/TA by attacking the REE (only if third-party extensions offering advanced protections are present along with default OP-TEE build). To achieve these objectives, we assume an adversary that ① has physical access to the device, ② can execute code on REE side, and ③ whose privileges are confined to Exception Level 0 (EL0) in the normal world.

B. Installing a malicious TA through data bus faults

All TAs in the TEE are signed by the OEM's private key (which is not present on the device post-deployment) and signature verification happens every time a TA is loaded. Installing a malicious TA without fault injection then becomes the equivalent of forging digital signatures. With fault injection, one logical attack strategy is to use an *instruction skip* to bypass this signature verification. However, instruction skips are not viable because of the presence of control-flow integrity checks. Consequently, we use data bus faults and exploit the programming convention of using non-zero integers to represent errors and a `0x0` to represent success. We use data bus faults to *convert* a non-zero return value of a function to `0x0`, which then tricks TA signature verification to load a malicious TA.

We now explain in detail how attack works. OP-TEE follows an offline signing and online verification process. Since the signature keys used to sign the TAs are not present on-device, forging signatures through signing key leakage is impossible. Therefore, the only possible way to load a self-signed TA is to force a failure of this verification step. We observed that while several error states (like security errors, out-of-memory errors, etc.) were given 32-bit non-zero values, `TEE_SUCCESS` was given a value `0x0`. This is common as almost all Linux-based function calls conventionally use a `0` to denote success, and other integers denote several erroneous operations. We exploit this convention for the attack.

Fault attacks require an appropriate triggering mechanism that can drive the start and end of the active fault injection process. We use power-based side-channel traces of multiplications to trigger fault injection. Multiplications are computation-heavy operations that require higher power consumption than other operations, and such multiplications are heavily used in OP-TEE's RSA-based signature verification. We use a non-invasive triggering mechanism to indicate when our signal generators should begin sending EM pulses. The result of this attack is a self-signed adversarial controlled TA getting installed in the secure world of OP-TEE, thereby violating the security guarantees of ARM TrustZone.

C. Stealing encryption keys through address bus faults

Next, we utilize address bus faults to retrieve the symmetric key from third-party key management services, such as *SeCReT*, that effectively encrypt communication between a legitimate TA and a trusted CA. Our address bus faults are able to bypass *SeCReT*'s defences because our faults *force the CA's access to its own memory pages to be invalid/faulty*, and thus cause no violation of *SeCReT*'s threat model. In other words, *SeCReT* trusts the CA owning the symmetric session key to behave innocently when handling its own key, while *SeCReT* guards malicious accesses from other agents (including debuggers to trace the victim) in the system. However, because of the address bus faults, we are able to force the CA itself to incur faults. This causes successful creation of coredumps with the symmetric session key (shared between CA/TA) accessible to an adversary.

IV. UUID confusion: a GlobalPlatform API specification vulnerability

GlobalPlatform API specification directs that ① each TA has its own UUID (Universally Unique Identifier) and ② this UUID is public knowledge. UUIDs allow a CA in the normal world to initiate communication with a TA in the secure world by uniquely labeling all TAs. Interestingly, the specification does not mention how such a situation should be handled in case two TAs are assigned the same UUID. However, with our bus faults, the adversary can not just deploy a malicious TA, but also choose an arbitrary UUID for it. The adversary can install its malicious TA using the bus fault attack, which can then masquerade as any innocent pre-installed TA. UUID confusion has a far-reaching consequence for an in-the-wild deployment of OP-TEE. As the adversary is free to choose any UUID of its choice and all legitimate (and whitelisted) UUIDs are public, it can masquerade as a legitimate TA and hijack all communication with innocent CAs. Furthermore, it can masquerade as *gatekeeper* TA and whitelist all incoming connection requests from any CA, thereby effectively removing any source authentication from the system.

V. Conclusion

Presence of countermeasures like access control checks, control flow integrity checks etc. complicate the portability of Fault Injection attacks relevant to FPGAs and ASICs to modern System-on-a-Chip (SoC). In this work, we mount a novel attack on a previously unexplored architectural aspect of an SoC- the system bus. We show how targeted electromagnetic injections during operation of memory instructions cause faults in the contents of data and address buses. To further contextualize the potency of bus faults, we mount an end-to-end attack on a practical implementation of ARM TrustZone, demonstrating a complete breakdown of TrustZone's security guarantees. This work, therefore, reinforces the importance of considering not only the cryptographic implementations but also the execution environment where the algorithms are expected to operate.

Speaker Biography

Anirban Chakraborty is a postdoctoral researcher at the Max Planck Institute for Security and Privacy, Germany and a visiting researcher at Ruhr University Bochum. He obtained his Ph.D from the Indian Institute of Technology Kharagpur, India, in 2024. His research interests broadly lie in the field of microarchitecture and systems security, trusted execution environments, and secure systems.