Attested Trusted Execution Environments for Transactional Workloads

Sid Hussmann
Gapfruit Technologies
Switzerland

July 13, 2021

Abstract

We present a generic TEE toolkit applicable for many use-cases in finance, healthcare or government. Gapfruit TEE is built on very strong security properties [1]. It embodies a microkernel operating system with capability-based security. Each component of the system is strongly isolated and has only access to the resources and services it really needs. The trust graph of each component is concisely defined and verified during build, deployment and run time. Gapfruit TEE runs on ARM, x86 and RISC-V leveraging the hardware security properties these chips provide. Different application runtimes and a simple API make it easy to integrate Gapfruit TEE into existing IT and OT infrastructure.

1 Introduction

A hosted computing provider may guarantee a certain amount of liability, to encourage users to run their software in the providers computing environment. IT security threats and vulnerabilities are rising exponentially. By offloading sensitive workloads to trustworthy computing environments, users can counter these threats. However, existing TEE technologies fail to address many rudimentary requirements such as ease-of-use, isolation, attestation, management and scalability.

This paper has two main, but distinct audiences: Product manufacturers and system integrators.

- Product manufacturers, such as HSM vendors, use *Gapfruit Trustworthy Execution Platform* (TEP) [1] to build secure products, like TEE appliances [2].
- System integrators integrate such TEE appliances into their infrastructure using the TEE API [5]. The integration of Gapfruit TEEs is similar to the ease-of-use of server-less computing, combined with attestable security.

The expression *Trusted Execution Environment* (TEE) is quite ambiguous. The term is used in different sub-domains in the computing ecosystem. Though, there is one common characteristic: The main goal is to protect trusted code and data during runtime. Most TEEs on the market distinguish between trusted and untrusted code, whereas the trusted code generally has more access to the overall system. However, in practice, it is not so straight forward to divide applications into strictly separate groups, trusted or untrusted, good or malicious.

This is where Gapfruit TEE shines. The capability-based architecture of Gapfruit allows us to govern the trust relationship of each sub-system down to the hardware. Combined with strong isolation and attestation, this technology forms the basis of truly trustworthy computing.

For device and appliance manufacturers, the Gapfruit TEE system can be customized and integrated in their products. For IT or OT system architects, products running Gapfruit TEE, are easily accessible from their infrastructure, with an open and simple API [5]. Existing applications run inside a Gapfruit TEE without or with minimal modifications.

This white-paper describes the high-level concepts of the *Gapfruit Trusted Execution Environment*. An in-depth discussion about the core technology can be found in the technical white-paper [1].

2 Vision

While some early design decisions of Unix led to the insecurity of today's operating systems [1], one thing is quite beautiful: The simplicity of the interface between processes. The term "do one thing and do it right" combined with the simplicity of chaining things together, is one of the reasons why Unix/Linux is so popular. Figure 1 shows how you interact with an application on Unix. After spawning the process, you send input to *stdin* and listen to output of *stdout*. Any side information about the computing that helps to analyze what's going on are being written to *stderr*. When the computation is done, the process sends an *exit code* that indicates success or a specific error, while the number Zero meaning success.

Figure 1: The Simplicity of Unix Processes

What if we could have all the benefits of this simplicity, combined with attested strong security guarantees? The application developer would not need to deal with any form of attestation. They would develop and build the application on their favorite operating system. Run it. Test it. If it works deploy it to a *Trusted Execution Environment* (TEE) 2. The application would not even know it runs within a TEE. Even the people who use these protected applications would not need to fully understand all the concepts around TEEs.

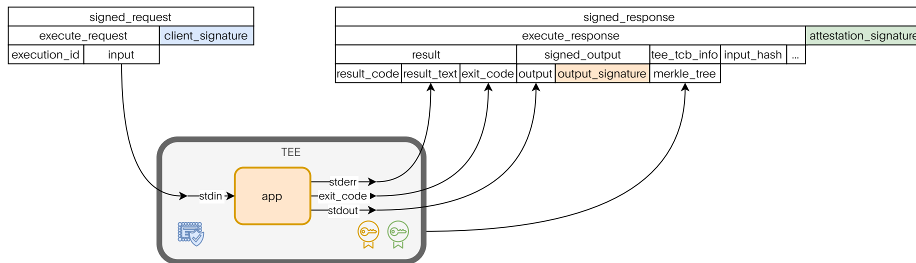

Figure 2: The Simplicity of Unix Processes in TEEs

The concept of Gapfruit TEE has similarities with concepts of *Serverless Computing* and *Functional Programming*. The relation to *Serverless Computing* lies in the fact that you don't have to deal with any operating systems, virtual machines, or container runtimes. You just build, deploy, and run your application. The Gapfruit TEE provides confidence in the absence of functional impurity, while the full transitive closure of the Trusted Computing Base (TCB) is measured. In this regard, it is related to some ideas of *Functional Programming*. Gapfruit TEEs go way beyond that. The technology combines this simplicity with the strong security guarantees described in the following section.

3 Technology

The *Gapfruit Trusted Execution Environment* technology is built on top of a microkernel operating system with capability-based security [14]. The *Trusted Computing Base* (TCB) is extremely small and can be verified during build, deployment and run time.

From our experience in finance and government, *Trusted Execution Environments* should provide at least the following properties:

- Physically and logically secure the execution of the application code from any interference and leakage (Isolation Property).
- Attest to a verifier the runtime integrity of an application, including the whole TCB.
- Creating an immutable record of the execution of the application code, detailing input, output, time, and device state (Audit Property).
- Restrict access to start an execution to authorized entities (Authorization Property).

This entails having a mechanism to securely load code, protect code from alteration, and extend to protecting the processed data and its output. A Gapfruit TEE can prove that a certain output was generated from a specific input, executed at a particular time with specific code. So it works like a notary that proves its trustworthiness through attestation.

The building blocks in a Gapfruit system are called SLICEs. A SLICE is a *Secure and Light Instance of Contained Enclave* which has strong isolation guarantees so that application and data are protected at runtime. Furthermore, potentially malicious code is contained from breaking out. An analogy would be the objectives of enclaves combined with what virtual machines or sandboxes try to achieve (Figure 3). So the isolation of SLICEs is guaranteed from outside-in as well as from inside-out. This is important, as it is not always clear what stakeholder considers which component of a system as trustworthy.

Each SLICE only receives access to the resources and services it really needs. The desired state of the SLICE topology is defined in a nested configuration mechanism, which forms a mandatory access control system [6].

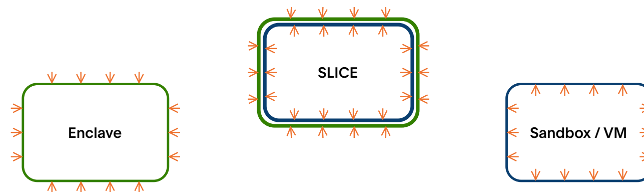

Figure 3: The isolation of a SLICE in contrast to enclaves and sandboxes

Apart from the strong isolation, another core concept of Gapfruit is how dependencies are governed.

The first type of dependency is *Resource Distribution*. A child component depends on its parent. Each dependee is designed as simple as possible, so it can

be verified for correctness. At the root of this dependency tree lies the micro-kernel. A parent component provides its children with resources and establishes service connections to other components.

These connections form the second type of dependency: *Service Topology*. SLICEs are connected with each other via a service-oriented architecture where a client depends on a server providing a service.

The third type of dependency controls the supply chain of *Software Dependencies*. Where a SLICE depends on binaries, libraries, or other artifacts that are part of distributable packages. Figure 4 shows these three types of dependencies in three views of the *same* system.

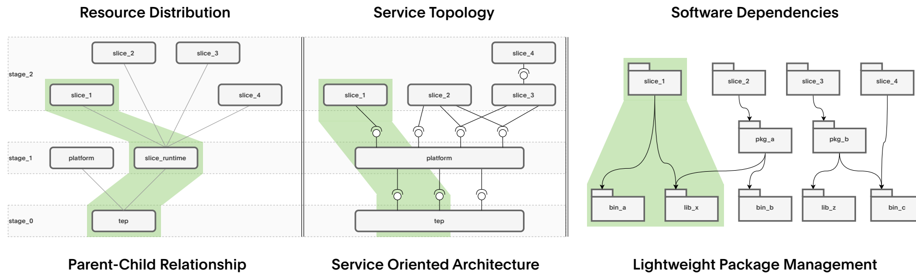

Figure 4: Topologies of dependencies

3.1 Attested Transactional Computation

With these two core concepts, strong isolation and control over dependencies, we create the TEE scenario.

Figure 5 shows a high-level overview where a TEE client connects to a Gapfruit TEE appliance via gRPC [7]. This TEE client sends requests and receives responses. If the request is of type *ExecuteRequest*, the application within the TEE gets executed.

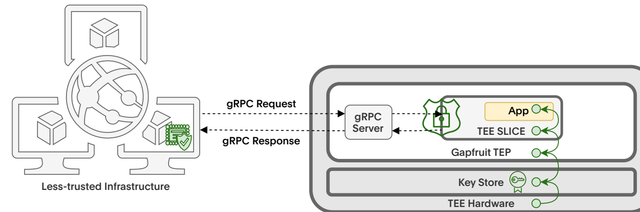

Figure 5: Remote trusted execution

An internal view of the *Trusted Execution Environment* appliance in form of *Service Topology* can be seen in figure 6. The *gRPC Server* on the left is connected to a *NIC* session (Network Interface Card). This way it can be reached over an external network from a TEE client, seen in Figure 5. The *gRPC Server* contains complex third party code [7]. Multiple threads are running, multiple TCP connections and TLS sessions are handled, as well as HTTP and Protobuf [8] is being parsed. The system is designed in a way that the *gRPC Server* is outside of the TCB and can be restarted at any time without having

an impact on the health of the system. The integrity is protected within the signed requests and the signed responses.

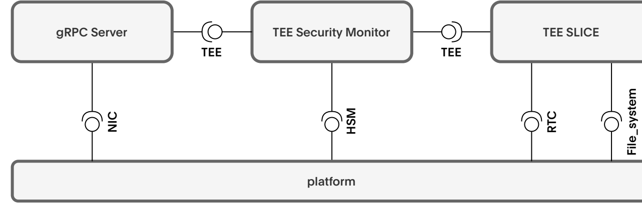

Figure 6: Customizable Trusted Execution Environments

The *gRPC Server* forwards requests to the *Security Monitor*. The *Security Monitor* has two main responsibilities: Enforcing access control for TEE requests and providing attestation information to responses. After verifying the client-signature, the *Security Monitor* forwards the request to the *TEE SLICE*. The *TEE SLICE* loads the executable, calculates its hash, and executes it in the runtime given by the application type. The *TEE SLICE* then provides input from the request to the executable. After execution, the *TEE SLICE* creates a response containing the output from the executable and sends it to the *Security Monitor*. Here, the response is enriched with information about the system, such as the platform state in form of a Merkle tree [11], the current timestamp, and secure counter value for non-repudiation. The response is then signed with an attestation key and sent back to the TEE client via the *gRPC Server*.

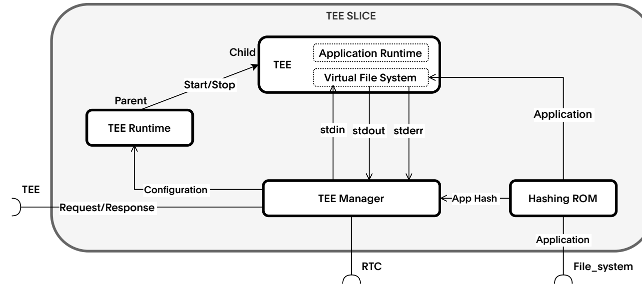

Figure 7: Trusted Execution Environment Internals

Figure 7 shows the internals of a *TEE SLICE*, introduced in Figure 6. The *TEE Manager* component receives requests and sends back responses from and to the *Security Monitor* accordingly. The *TEE Manager* controls the *TEE Runtime* and establishes communication channels to the application's *stdin*, *stderr*, and *stdout*. The *TEE Runtime* is a parent component that spawns the *Application Runtime*, e.g., a full JVM [3] or WebAssembly runtime [4]. The *Hashing ROM* component loads the application image from the file-system, calculates its hash, and provides said hash value to the *TEE Manager*. Accordingly, the *TEE Manager* includes said hash value into the execution record of the respective application code. The *Hashing ROM* then provides the application code as read-only memory to the *Application Runtime* to execute. The *Hashing ROM* and its relationship to the *Application Runtime* are built in a way so that time-of-check to time-of-use (TOC/TOU) attacks are prevented. After successful

execution, the execution record and the output on *stdout* as well as *stderr* are added to the response, which is sent back to the *Security Monitor*.

3.2 Attestation

The chain of trust used in software attestation is rooted at a signing key owned by the hardware manufacturer of an HSM, TPM [9], or any other form of a secure element such as OpenTitan [10]. The architecture in Figure 6 shows that only the *Security Monitor* has access to the *HSM* service. This is enforced by the capability-based security of Gapfruit TEP. The platform makes sure that only the *Security Monitor* of a specific domain has access to the key store of the respective domain. Every gRPC response is signed with a domain-specific attestation key. The certificate of the attestation key is signed by a device-specific key, which in turn is signed with a root key of the device manufacturer.

Since domain-specific keys are linked to device keys, any response that is signed by the respective *Security Monitor* can be cryptographically linked to a specific device via the certificate chain of the attestation key. A gRPC client may request a quote from a domain. The quote can be used to provision a TEE.

The quote holds a view of the complete *Trusted Computing Base* in form of a Merkle tree [11]. The Merkle tree represents the TCB of the whole subsystem of the device which is involved to execute code inside of a *TEE SLICE*. It unifies the TCB in terms of *Resource Distribution*, *Service Topology,* and *Software Dependencies* as shown in Figure 4. It provides confidence in the absence of functional impurity. While up to *Stage 0* the integrity of the system is enforced with the static root of trust measurement (SRTM), the Merkle tree represents the dynamic root of trust (DRTM), which in turn is linked to *Stage 0*. The full expansion of state of the TEE is measured and part of the Merkle tree.

3.3 Customizable TEEs

Section 3.1 described how a minimal TEE system can be designed. Figure 8 shows a simple system combining three SLICEs: A *gRPC Server*, a *Security Monitor* and a *TEE SLICE*. This may suffice for certain use-cases. Due to the nature of the Gapfruit TEE framework, we are a lot more flexible. A more detailed diagram of what it means to have customizable *Trusted Execution Environments* can be seen in figure 8. This example shows a multi-tenant situation where *Domain Orange* belongs to one tenant, while *Domain Green* belongs to another. Also, several instances of *TEE SLICE*s can be deployed within each domain, so TEE applications can run in parallel.

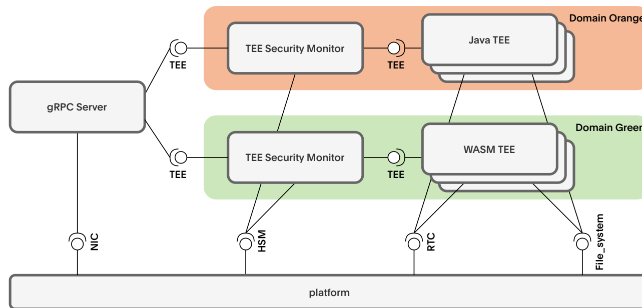

Figure 8: Customizable Trusted Execution Environments

In another scenario, Gapfruit TEE runs within the *Secure World* of ARM Trustzone [13]. In this case the *gRPC Server* would be replaced with a component that receives API calls from the *Normal World*. Since Gapfruit supports virtualization, the same mechanism can be applied to access the TEE API from within virtual machines.

4 Use-Cases

We deployed Gapfruit TEEs successfully on Securosys hardware [2] with the same protection against physical attacks as their HSMs. The use-cases described in this section originate from real-world scenarios in the finance sector.

4.1 Compliance Verification

Many areas in our life are automated. Especially the banking sector was one of the early adopters of IT automation. The things that currently aren't automated are typically things human individuals have to vouch for. Meet Alice, an enthusiastic, highly paid banking professional. In this simple example she is the one who has to approve a business transaction. She has to go through a compliance checklist, answering questions regarding the business transaction details: Is the amount within the threshold? Is the money going to a sanctioned company or country? In case of Bitcoin (BTC), is the current exchange rate within bounds? Does the Know-Your-Customer (KYC) report comply? Is the BTC address white- or blacklisted? After filling out the compliance checklist, she signs that piece of paper with a pen and adds it to the stack of compliance reports, which forms the audit trail. After verifying that the business transaction meets the compliance requirements, she authenticates herself to the HSM and approves the transaction with her private key. The HSM then signs the business transaction with its internal private key.

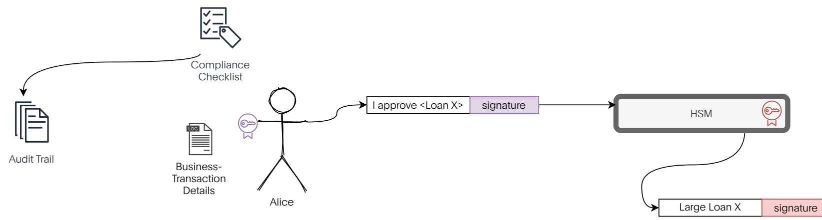

Figure 9: Manual Compliance Verification

Now what is stopping us from automating this process using conventional banking infrastructure?

Today's general-purpose operating systems such as Windows, Linux, *BSD or OSX have millions of lines of code (LOC). Linux currently counts 32M LOC and the Windows kernel is estimated to consist of 65M LOC [1]. All of which evolved over decades and their initial focus was on usability, performance, and portability. Different security mechanisms were considered after the architecture was set in stone. There is no way to guarantee the integrity of the system. Thus, there is no way to cryptographically prove that all the important aspects of a business transaction were considered.

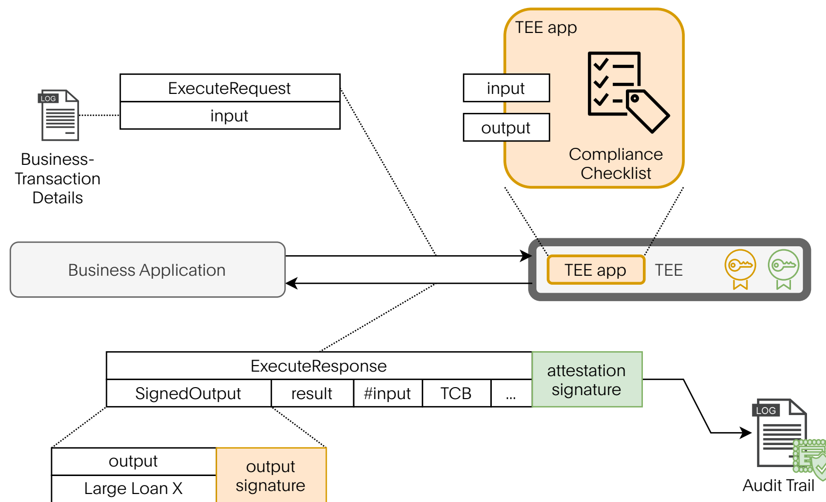

Figure 10: Automated Compliance Verification

With Gapfruit TEE, automation of this critical part of banking becomes possible. Figure 10 shows how such an infrastructure looks like. The Business Application (E.g. the core banking system), creates an *ExecuteRequest* with an input that contains the business transaction details. This request is then sent to the TEE appliance. The compliance checklist and the code that verifies if the business transaction details meet the compliance requirements, are part of the *TEE app*. The *TEE app* returns its decision as part of the output and result. The TEE appliance then cryptographically signs the *output* as part of the *SignedOutput* and creates an *ExecuteResponse*, which is signed with an attestation key. This approach with the two signatures has two main benefits.

For one, the *SignedOutput* can be directly used as signed transaction, just like it would have come from a HSM (see Figure 9). For the other, the signed *ExecuteResponse* proves that the output was generated from a specific input, executed at a particular time with specific code. This proof is added to a database as part of the audit trail.

4.2 Approval Framework

The example above illustrates how the Gapfruit TEE can be used to automate compliance verification.

For extra security of even higher business transactions, Figure 11 shows a real-world scenario, similar to how it is deployed in a banking environment. In this example a large business transaction needs to be authorized. The architecture has several entities:

- *Core Banking Infrastructure*: This is where all the business logic of a bank is computed.
- *HSM*: The Hardware Security Module that signs large transactions, interbank-clearance, smart contracts, etc.
- *Alice and Bob*: Bank employees that need to approve certain transactions.
- *TEE*: Trusted Execution Environment as physical device with the same protection against physical attacks as the HSM. Executes the *TEE app* and attests the security properties of the device.
- *TEE app*: The application that decides about the approval of the business transaction based on the same information provided to Alice and Bob. The *TEE app* is developed by either the bank or a 3rd party, and is verified and deployed to the device by the *Security Officer*.
- *ExecuteRequest and ExecuteResponse*: Protobuf messages defined in the API [5].

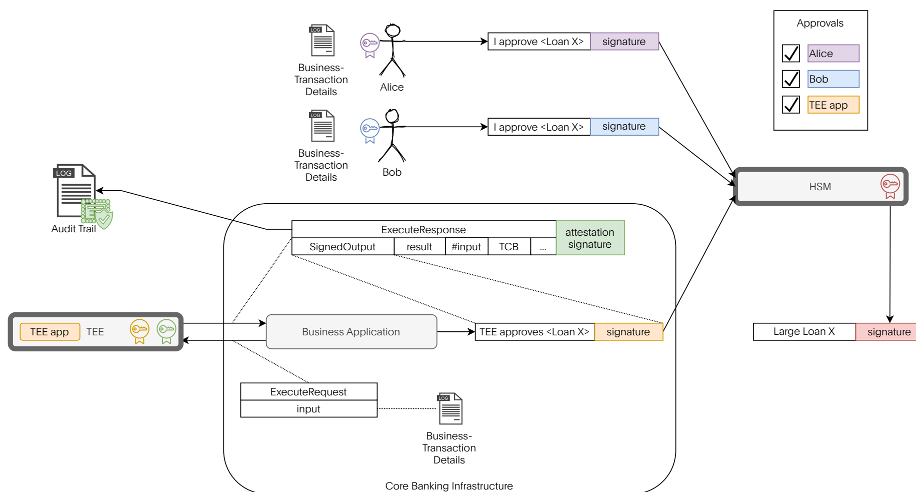

Figure 11: Multi-Authorization Use-Case

The HSM appliance on the right has a multi-authorization feature [12] that signs a large business transaction, when three parties approve: Alice, Bob, and a TEE application. For this, a security officer has to onboard each of the approver's public key to the HSM. On the left, there is an appliance shown that runs Gapfruit TEP with the TEE functionality. During provisioning, an *attestation key* and an *output signing key* is generated. Also, the TEE application is loaded. In this case, the TEE application is a compliance filter that goes through a series of checks that make sure the business transaction satisfies the bank's interests and regulations. Details about the to-be-checked business transactions are part of the *input* of the *ExecuteRequest*. The *Business Application* within the *Core Banking Infrastructure* sends such an *ExecuteRequest* the TEE, which provides the input to the TEE application, which in turn executes the compliance checks over said input. If the checks pass, the TEE application writes the output which then gets signed with the *output signing key*. This signed output is then sent back as part of the *ExecuteResponse*. Exactly this signed output is then sent to the HSM as the final approval for the business transaction. Additionally, the *ExecuteResponse* is signed with the *attestation key* and then added to an audit database so it can be verified by internal or governmental regulatory agencies. The *ExecuteResponse* attests that a certain output was generated from a specific input, executed at a specific time with specific code.

4.3 Confidential Computing

Gapfruit TEE allows companies to collaborate without exposing their private data to each other, such as bank account details, to determine and detect money laundering patterns. A machine learning model would be deployed to the TEE, while the data sent from the different banks are encrypted with the public key, whose private key is only accessible from within the TEE. This helps banks operate better in terms of money laundering challenges in their KYC processes and reduces false positives without exposing confidential data to the competing banks.

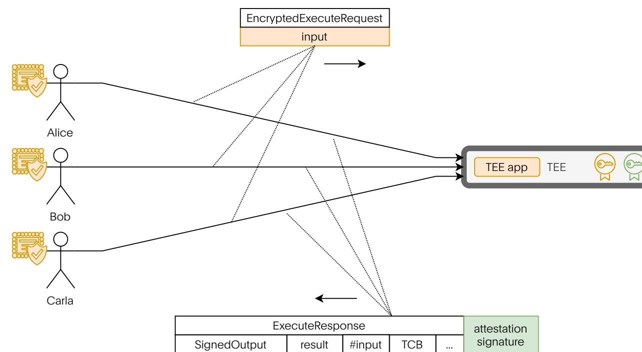

Figure 12: Confidential Computing

There are countless numbers of use-cases involved in confidential computing:

- Median Interest Rate
- DNA Risk Group

- Credit Qualifications
- Loan Fulfillment
- Market-rate Calculations
- Privacy-preserving Analytics

Outsourcing all kinds of data processing to external cloud infrastructures has become best practice. Still, a lot of companies don't trust cloud providers with their data. With the confidential computing properties of Gapfruit TEE, you can design a system, where most of your infrastructure runs within the cloud, while having control over the most sensitive data on prem, running within a TEE - never expose any confidential data to the cloud provider.

4.4 Database Transaction Validation

In cases where distributed blockchains are too slow, Gapfruit TEE may be used to execute call-backs during a database transactions, which would lead to verifiable, integrity protected database entries.

About Gapfruit AG

Gapfruit is a deep-tech company based in Switzerland with a proven track record in systems security, software engineering, and innovation. The founding team developed a military-grade operating system fulfilling the requirements set by national governments and security agencies across the world for ironclad security. With this expertise, Gapfruit brings scientifically recognized academic research to real-world products for today's and future challenges. The developers at Gapfruit have been contributing to the Genode Framework [14] for many years.

If you want to deliver trustworthy products yet focus on your core expertise, get in contact with us today.

info@gapfruit.com

Abbreviations

	Meaning
AML	Anti Money Laundering
API	Application Programming Interface
App	Application
BTC	Bitcoin
DRTM	Dynamic Root of Trust Measurement
gRPC	Google Remote Procedure Call
HSM	Hardware Security Module
IT	Information Technology
JVM	Java Virtual Machine
KYC	Know Your Customer
LOC	Lines of Code
NIC	Network Interface Card
OT	Operational Technology
ROM	Read-only Memory
SLICE	Secure and Light Instance of Contained Enclave
SRTM	Static Root of Trust Measurement
stderr	Standard Error Interface of Unix Processes
stdin	Standard In Interface of Unix Processes
stdout	Standard Out Interface of Unix Processes
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
TEE app	Application that runs within the TEE
TEP	Gapfruit Trustworthy Execution Platform
TLS	Transport Layer Security
TOC	Time of Check
TOU	Time of Use
TPM	Trusted Platform Module
WASM	WebAssembly

References

[1] Sid Hussmann, Gapfruit TEP, Technical Whitepaper
https://www.gapfruit.com/technology

[2] Securosys Imunes
https://www.securosys.com/securosys-trusted-execution-environment-tee

[3] JVM Languages
https://en.wikipedia.org/wiki/List_of_JVM_languages

[4] Wasm
https://webassembly.org/

[5] Gapfruit TEE API
https://docs.gapfruit.com/api/tee_api.html

[6] Mandatory Access Control
https://en.wikipedia.org/wiki/Mandatory_access_control

[7] gRPC, a high performance, open-source universal RPC framework
https://github.com/grpc/grpc

[8] Protocol Buffers
https://developers.google.com/protocol-buffers/

[9] Trusted Platform Module
https://trustedcomputinggroup.org/resource/
tpm-library-specification/

[10] OpenTitan
https://opentitan.org/

[11] Merkle Tree
https://en.wikipedia.org/wiki/Merkle_tree

[12] Securosys, Transaction Security Broker
https://www.securosys.com/securosys-transaction-security-broker

[13] ARM TrustZone
https://developer.arm.com/ip-products/security-ip/trustzone

[14] Genode Framework
https://www.genode.org

[15] Jiewen Yao, Vincent Zimmer, Building Secure Firmware
https://link.springer.com/book/10.1007%2F978-1-4842-6106-4

Appendix

Threat Model

This section describes the threat model of TEE appliances built with Gapfruit TEP, such as [2]. The thread model is derived from the STRIDE model described in [15]. Mitigations for hardware-based attacks are important to get right. That's why we work closely together with hardware manufacturers such as HSM vendors. Since every hardware manufacturer has different techniques, we consider hardware-based attacks out of scope for this analysis.

Table 1: Threat for Asset - TEE Appliance

Threat	Example
Spoofing	s-1: The attacker may receive a legitimate request and execute it on a different device than the TEE appliance.
Tampering	t-1: The attacker may change settings of the TEE appliance.
	t-2: The attacker may change the TEE application or data during an execution.
	t-3: The attacker may load malicious code as TEE application to the TEE.
	t-4: The attacker may change parts of an *ExecuteRequest*.
	t-5: The attacker may change parts of an *ExecuteResponse* or *QuoteResponse*.
Repudiation	r-1: The attacker may try to change parts of an entry of the audit-trail.
	r-2: The attacker may delete an entry of the audit-trail.
	r-3: The attacker may prevent adding an entry to the audit-trail.
Information disclosure	i-1: The attacker may intercept gRPC network traffic.
	i-2: The attacker may try to access data of another TEE application from their TEE application.
	i-3: In an multi-party computation use-case, the attacker may try to access the input of another party.
Denial of service	d-1: The attacker may flood the gRPC server.
	d-2: The attacker may utilize a lot of resources within the TEE application such as RAM and/or CPU.
Elevation of privilege	e-1: If the attacker gains access to credentials to the Security Officer role, they can change settings and the TEE application.
	e-2: If the attacker gains access to credentials of the client, they can start executing a TEE application.

Table 2: Adversary for Asset - TEE Appliance

Adversary	Example
Network attacker	n-1: The attacker may connect to the system by network in order to eavesdrop, intercept, or modify the network packets.
	n-2: The attacker may connect to the system by network in order to exploit a vulnerability in the network stack or gRPC server.
	n-3: The attacker may connect to the system and send unauthorized *ExecuteRequest*.
Unprivileged software attacker	u-1: The attacker may hide malicious code within a TEE application that tries to escape the isolation and attack the platform or TEE applications from other security domains.
	u-2: The attacker may send input as part of an *ExecuteRequest* that triggers a vulnerability within the TEE application.
Privileged admin attacker	p-1: The attacker may try to load a TEE application into their security domain that tries to escape the isolation and attack the platform or TEE applications from other security domains.

Table 3: Mitigation for Asset - TEE Appliance

Mitigation	Example
Protection	t-2 Only authorized users are allowed to change the TEE application or data.
	t-2 Any change of the TCB during an ongoing execution will cancel the operation and return an error.
	t-4 Every *SignedRequest* that holds the *ExecuteRequest* is signed with a client key. Any change of parts of the request would invalidate the signature and the request will be dropped.
	i-1 Every gRPC session is protected with TLS.
	i-2 The virtual file-system that is provided to the TEE application is completely isolated from the rest of the system using capability-based security.
	i-3 Every input of an *EncryptedExecuteRequest* is encrypted with a public-key, whose private-key is only accessible within the physical device.
	d-2 Every TEE application has clearly enforced resource boundaries, which when exhausted would terminate an execution and return an error.
	n-1 Every gRPC session is protected with TLS and every gRPC message is signed.
	n-2 Since the network stack is outside of the kernel and only part of the isolated gRPC server, any exploit on the stack will only have an effect on the availability of the gRPC server. An exploit on the gRPC stack won't affect the integrity of any gRPC messages.
	n-3 Every authorized client certificate is specifically white-listed. Any *SignedRequest* with a certificate that isn't part of the allowed-clients list, gets dropped.
	u-1, p-1 Every TEE application is strongly isolated and the access to resources are governed by capability-based security.
	u-2 The attack would only affect their own TEE application due to the strong isolation and capability-based security.

Table 4: Mitigation for Asset - TEE Appliance

Mitigation	Example
Detection	s-1 Every *SignedResponse* is signed with an device-unique attestation key which is only accessible within the physical device.
	t-1 Any *ExecuteResponse* and *QuoteResponse* holds the full transitive closure of the TCB in form of Merkle tree.
	t-5 Every *SignedResponse* that holds the *ExecuteResponse* or *QuoteResponse* is signed with the attestation key. Any change of parts of the response would invalidate the signature.
	r-1 Every *SignedResponse* that may be used for the audit-trail, is signed the attestation key. Any change of parts of the response would invalidate the signature.
	r-2 If the optional secure monotonic counter is enabled, an auditor would notice a missing entry in the audit-trail.
	r-3 If the optional secure monotonic counter is enabled, an auditor would notice a missing entry in the audit-trail.
	d-1 Every SLICE of Gapfruit TEE is monitored for its health.
	e-1 Any *ExecuteResponse* and *QuoteResponse* holds the full transitive closure of the TCB in form of Merkle tree. An auditor can detect when the TCB has changed, and which specific part of the system. Also, any action on the system is logged.
Recovery	d-1 The gRPC server SLICE is designed in a way that it does not provide any services so it can be restarted when it becomes unresponsive without having impact on the rest of the system.
	e-1 The credentials for changing the configuration and the TEE application can be updated at any time. Or the device can be reset.
	e-2 The list of allowed client certificates can be updated at any time.