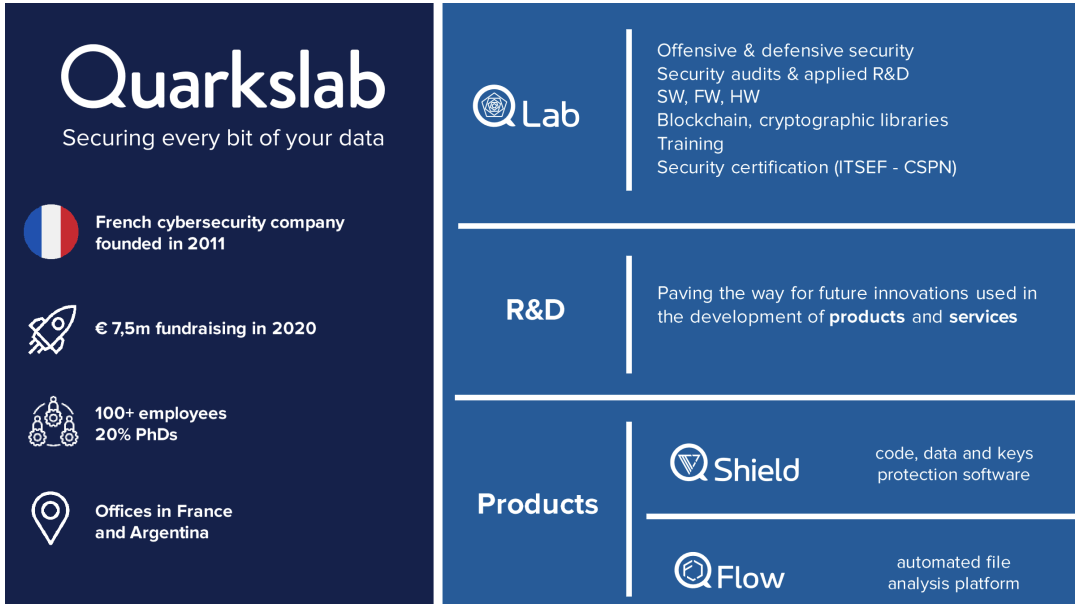
Confidential computing

Intel SGX assessment methodology

Motivation

Who are we?


Quarkslab
Securing every bit of your data
French cybersecurity company founded in 2011
€ 7,5m fundraising in 2020
100+ employees
20% PhDs
Offices in France and Argentina
Lab
Offensive & defensive security
Security audits & applied R&D
SW, FW, HW
Blockchain, cryptographic libraries
Training
Security certification (ITSEF - CSPN)
R&D
Paving the way for future innovations used in the development of products and services
Products
Shield
code, data and keys protection software
Flow
automated file analysis platform

Why this talk?

- Give some insights about the state of mind of an auditor attacking a SGX enclave
- Give some knowledge about the internals of SGX remote attestation

Confidential computing

Confidential computing

> *Confidential computing is a security and privacy-enhancing computational technique focused on protecting data in use*
> *The technology protects data in use by performing computations in a hardware-based trusted execution environment (TEE).*

Source: *https://en.wikipedia.org/wiki/Confidential_computing*

Trusted execution environments (TEEs)

> *Trusted execution environments (TEEs) "prevent unauthorized access or modification of applications and data while they are in use, thereby increasing the security level of organizations that manage sensitive and regulated data".*
> *Trusted execution environments can be instantiated on a computer's processing components such as a central processing unit (CPU) or a graphics processing unit (GPU)*

Source: *https://en.wikipedia.org/wiki/Confidential_computing*

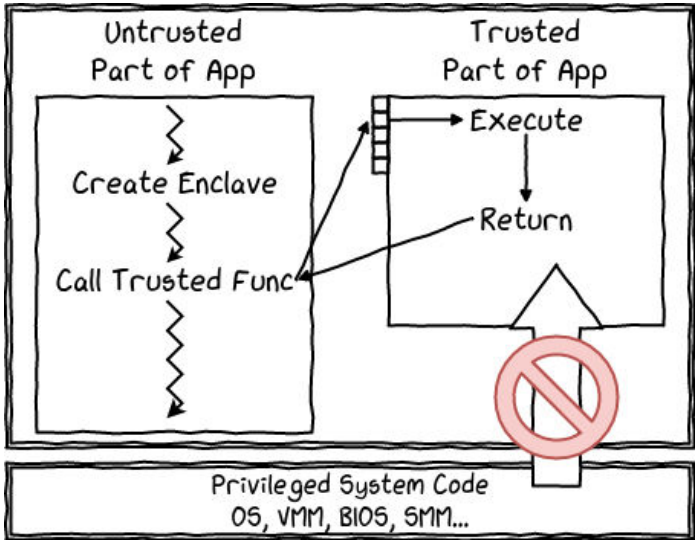
SGX

SGX

- Intel Software Guard Extension (SGX) was introduced in 2015 with microprocessors based on the Skylake micro-architectural family (6th generation).
- It is a set of operation codes to provide integrity and confidentiality for secure computing on a computer, where privileged software can be supposed malicious (kernel, hypervisor, etc.).
- In order to provide isolated execution, one of the main features upon which Intel relies is **software attestation**.
- This allows us to prove that communications between the users are made with the proper piece of trusted hardware, called **enclaves**.

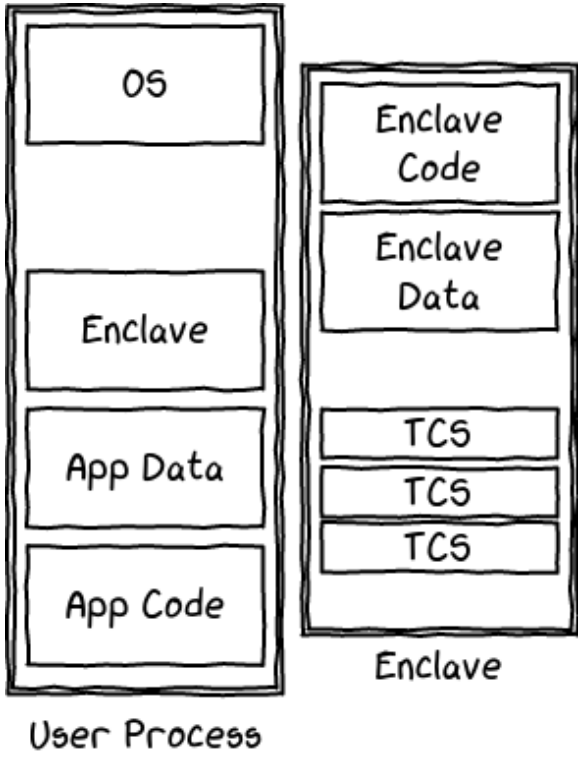
Enclaves

- The implementation of Intel SGX can be summarized in a few points:
  - An application is split into two parts: a **secure** one and a **non-secure** one
  - The application launches the enclave, which is placed in protected memory
  - When an enclave function is called, only the code within the enclave can see its data,
  - External accesses are always denied
  - When it returns, enclave data stays in the protected memory



Enclaves

- The secure execution environment is part of the host process, which means that:
  - The application contains its own code, data, and enclave
  - The enclave contains **its own code and its own data** too
  - SGX protects the confidentiality and integrity of the enclave code and data
  - Enclave entry points are pre-defined during compilation
  - An enclave can access its application's memory, but not the other way around



But SGX is deprecated?

- Due to the wide number of attacks on Intel SGX via side-channel attacks, Intel SGX solutions have been declared deprecated for the 11th and 12th generation of processors.
- Intel continues development for the Xeon family where the usage of the secure enclave should only be made in cloud and enterprise scenarios, namely when the attacker cannot get direct access to the devices.

Cache attacks

Controlled-Channel	Memory access monitoring via page fault	2015
Stealthy Page Table-Based	Observe enclave page accesses by exploiting the address translation process.	2017
SGX-Step	Configure APIC timers and tracks page table entries directly from user space	2017
Nemesis	Leak micro-architectural instruction timings from enclaved execution environments	2018
Off-Limits	Exploitation of the memory segmentation feature in 32-bit legacy mode	2018
Leaky Cauldron	Exploiting risk in memory management, from TLB to DRAM modules	2017
CopyCat	Controlled-channel attack that deterministically counts the number of instructions executed within a single enclave code page	2020

CacheZoom	Cache side-channel attack to virtually track all memory accesses of SGX enclaves	2017
Cache Attacks on Intel SGX	Access-driven cache-timing attack on AES when running in an Intel SGX enclave	2017
Malware Guard Extension	Prime+Probe cache attack on a co-located SGX enclave	2017
Software Grand Exposure	Cache attack on SGX enclave that is claimed easier to deploy and less detectable	2017
CacheQuote	Implementation weakness of the EPID protocol that leaks information via cache side-channel	2018
MemJam	Exploits false dependency of memory read-after-write	2017
FORESHADOW	Software micro-architectural attack to dismantle SGX's security objectives	2018

Speculative execution, branch prediction, fault injection

SgxPectre	SGX-variants of Spectre attacks	2019
Spectre Returns	Spectre-like attack that exploits return stack buffer instead of branch predictor unit	2018
SGXSpectre	Sample code demonstrating a Spectre-like attack against an Intel SGX enclave	2018
Fine-grained Control Flow	Reveals fine-grained control flows in an enclave with branch shadowing	2017
BranchCode	Infer the direction of an arbitrary conditional branch instruction	2018
Bluethunder	Bypass existing protection and reveal the secret information inside an enclave with a new pattern history table based side-channel attack	2019

RIDL	Leak arbitrary data across address spaces and privilege boundaries via speculative unprivileged and constrained attacks	2019
ZombieLoad	Meltdown-type attack that faults load instructions	2019
CacheOut	Speculative execution attacks to leak data from OS kernel	2021
SGXAxe	Follow-up of CacheOut	2021
CROSSTALK	Combination of micro-coded instructions for off-core requests and MDS to reveal internal CPU state	2021
Plundervolt	Exploitation of an undocumented Intel Core voltage scaling interface to corrupt integrity of SGX enclave computations	2020
VOLTpwn	Software-based attack that affects the integrity of computation on x86 platform by targeting a physical core to force non-recoverable hardware faults	2020

Software-based attacks

AsyncShock	Hijack enclave control flow or bypass access control by exploiting use-after-free and time-of-check-to-time-of-use	2016
Game of threads	Schedule asynchronous threads to bypass enclave protections	2020
SmashEx	Exploitation of asynchronous exceptions in SGX to expose enclave private memory and ROP attacks, without memory errors, side channels, or logic flaws	2021
A Tale of Two Worlds	Sanitization vulnerabilities at the ABI and API level to exploit memory safety and side-channel attacks in the compiled enclave	2019
The Guard's Dilemma	Two code-reuse attacks against enclave that provide full control of the CPU's general-purpose register without kernel privileges	2018
MicroScope	Micro-architectural replay attack by inducing pipeline flushes which can denoise the port contention channel of execution units	2020
Platypus	Software-based power side-channel attacks to exploit unprivileged access to the Intel Running Average Power Limit interface	2021

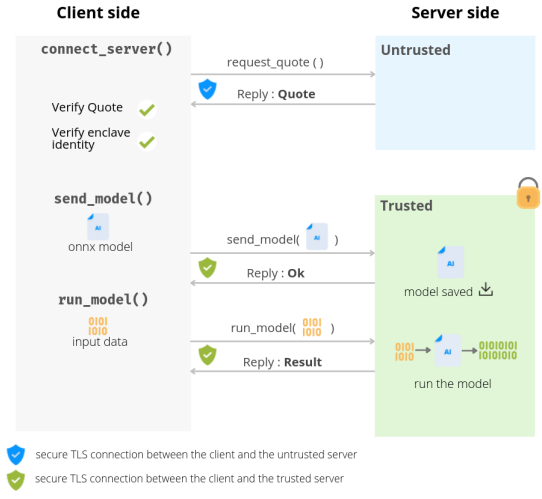
Mithril BlindAI

BlindAI

- BlindAI is an AI inference server with an added privacy layer that aims to protect the data sent to models.
- The main objective is to leverage state-of-the-art security to benefit from AI predictions, without putting users' data at risk.
- To that purpose, BlindAI is built on top of the hardware protection provided by Intel SGX, to provide end-to-end data protection.

BlindAI

- To interact with the enclave, BlindAI provides as well a Python-based client, which allows a user to upload, run, and delete models on the server, after establishing secure communication channels to the server via TLS.
- The server side is written in Rust and deployed on Azure VMs that have Intel SGX hardware.



AI inference

- The core building blocks in AI inference are the models used by the algorithm in order to recognize patterns or make decisions.
- The model is usually trained on a large dataset in order to obtain the most accurate values, which can be seen as a set of tensors and weights.
- The model training step is a complicated and costly step, highly dependent on the dataset used for training, and obtaining sound, precise models makes the difference between actors in the field.
- This is why holding the privacy of the models is essential for AI actors.

Audit

Audit

- The goal of the audit was to define the relevant threat models to the BlindAI, and to perform security testing based on the latter and within an allocated time frame.
- To that end, the following 4-step scope of work was defined and agreed upon by Mithril Security and Quarkslab:
  - Step 1: study of the state-of-the-art attacks on Intel SGX, and more broadly in the confidential computing area;
  - Step 2: definition of a threat model to refine a test plan relevant for the BlindAI;
  - Step 3: security assessment and testing of the BlindAI based on the previous steps;
  - Step 4: reporting and project management.

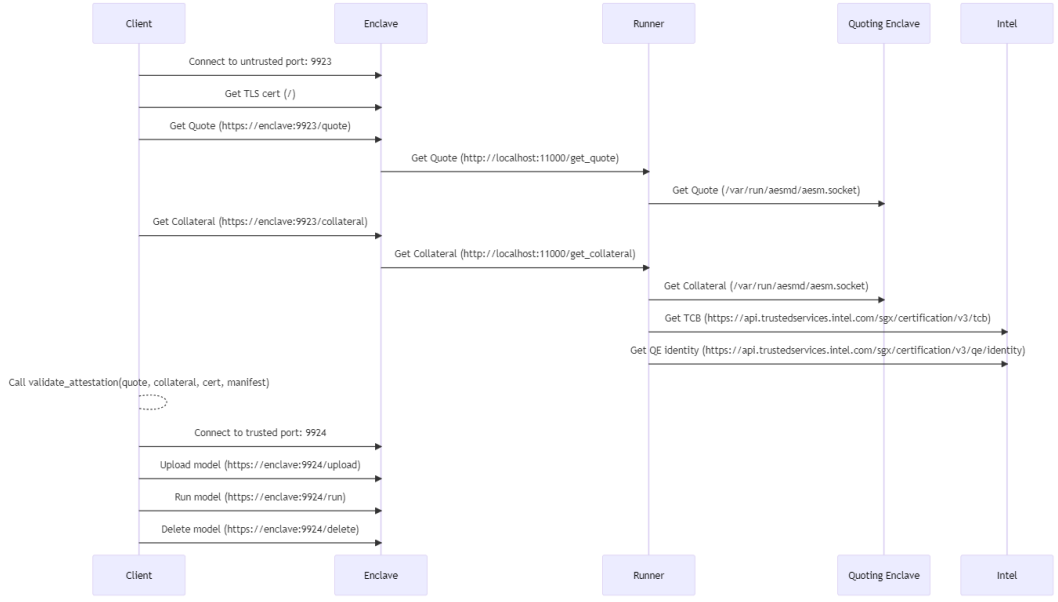
Audit

- The security audit was performed from January to March 2023 (40 person-days)
- Attacks on the client are considered only for Man-in-the-Middle attacks, considering an attacker on the server side.
- Other scenarios were considered out of scope of this assessment.
- Virtual machine details used during the security assessment:
  - Virtual machine (provided by Mithril): Azure VM DCs4v3
  - OS: Ubuntu 20.04
  - CPU: Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz

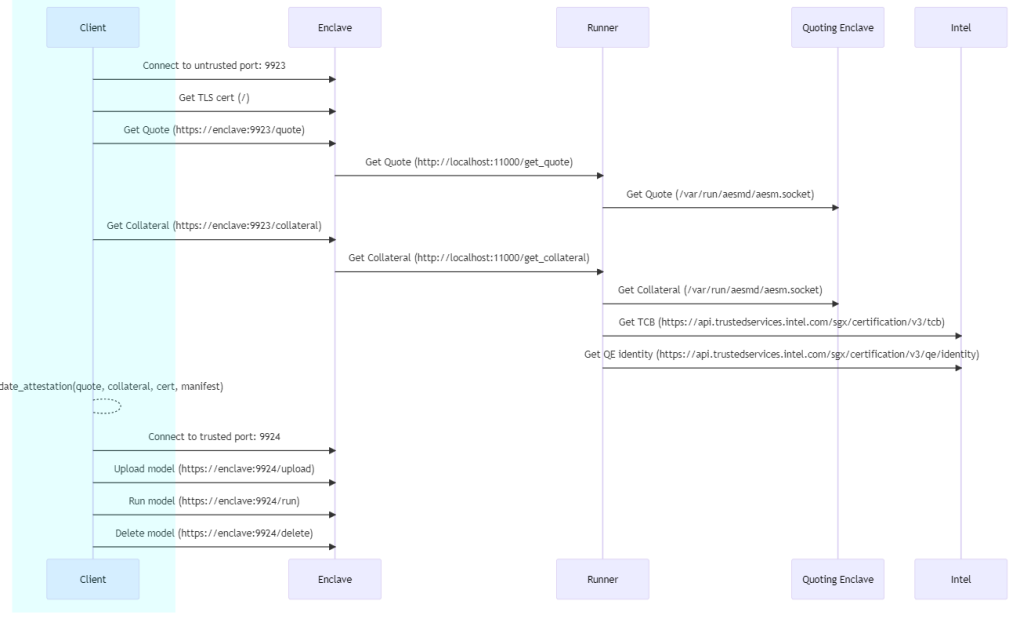
Methodology

- The taken approach and methodology can be summarized as a simple question:
  - As an attacker what do we control and what can we target?

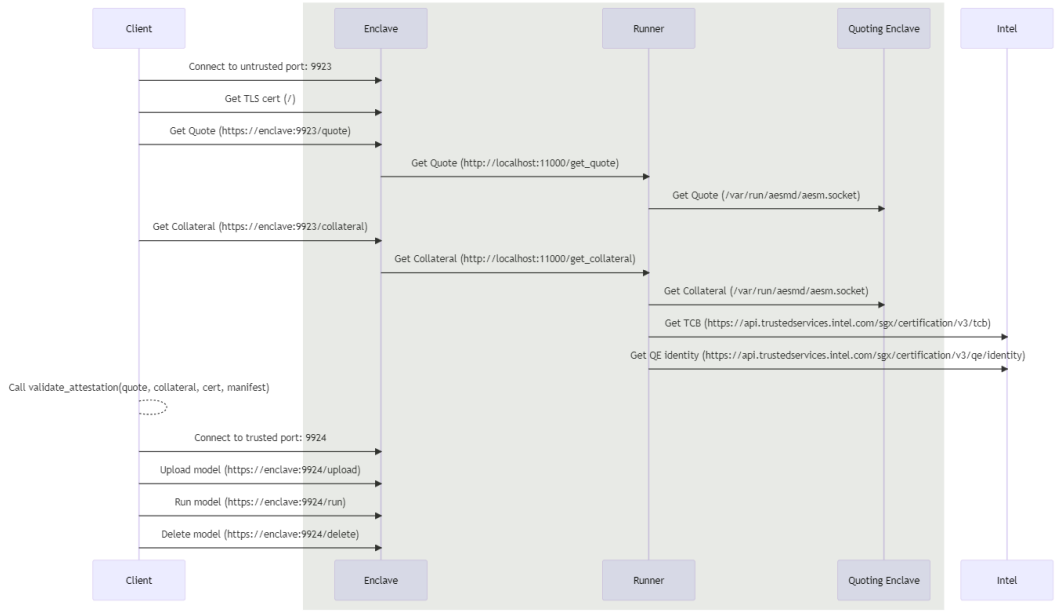
Cartography



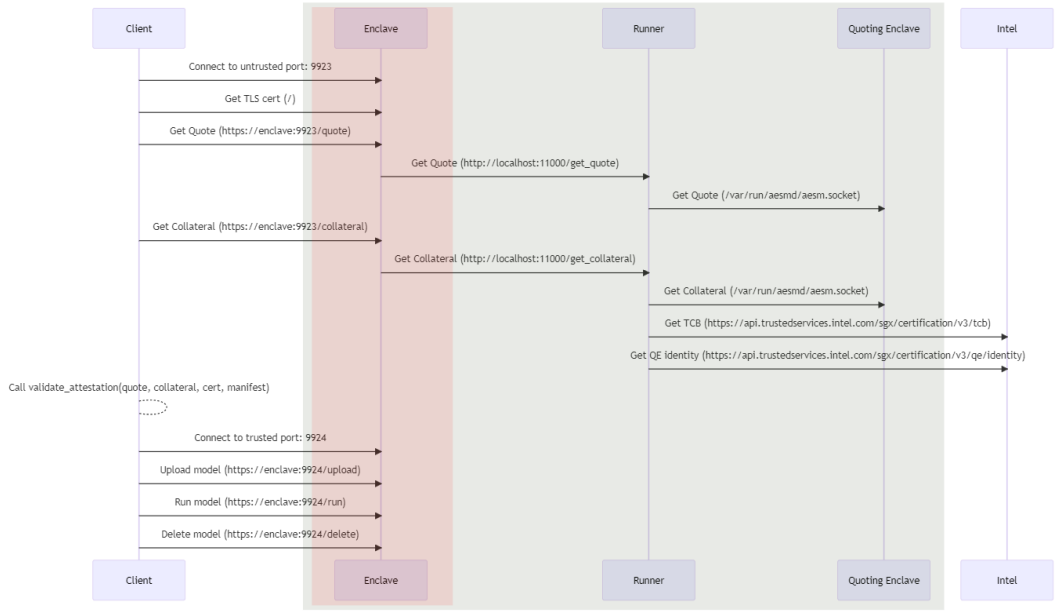
Cartography



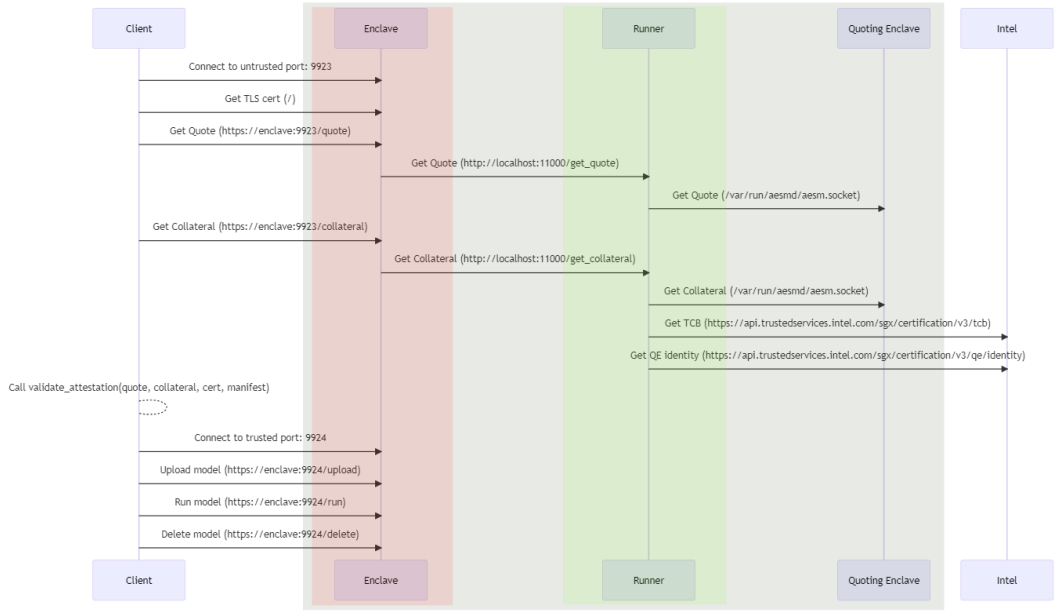
Cartography



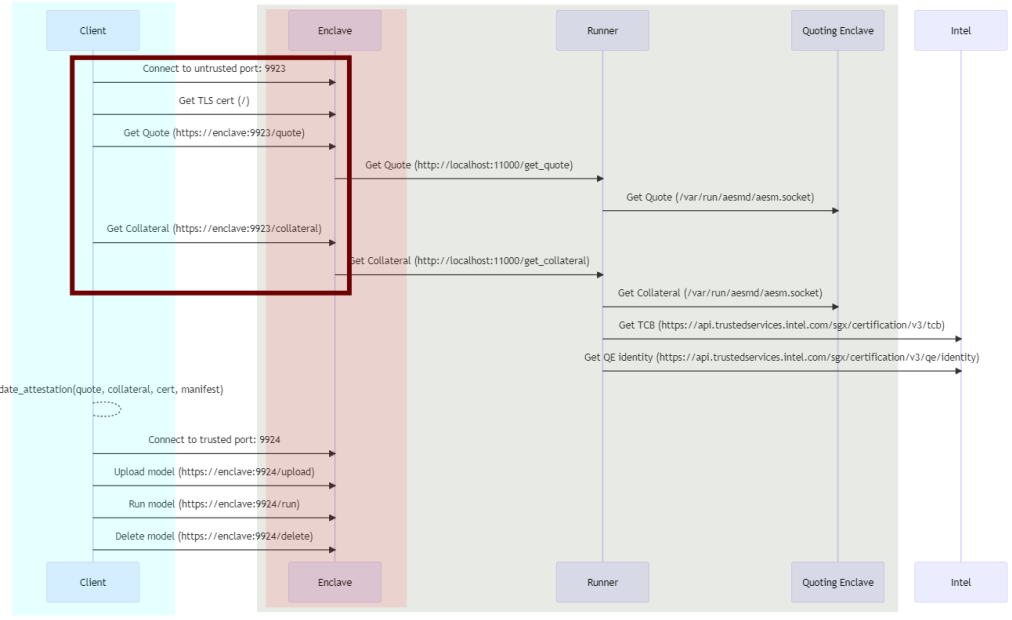
Cartography



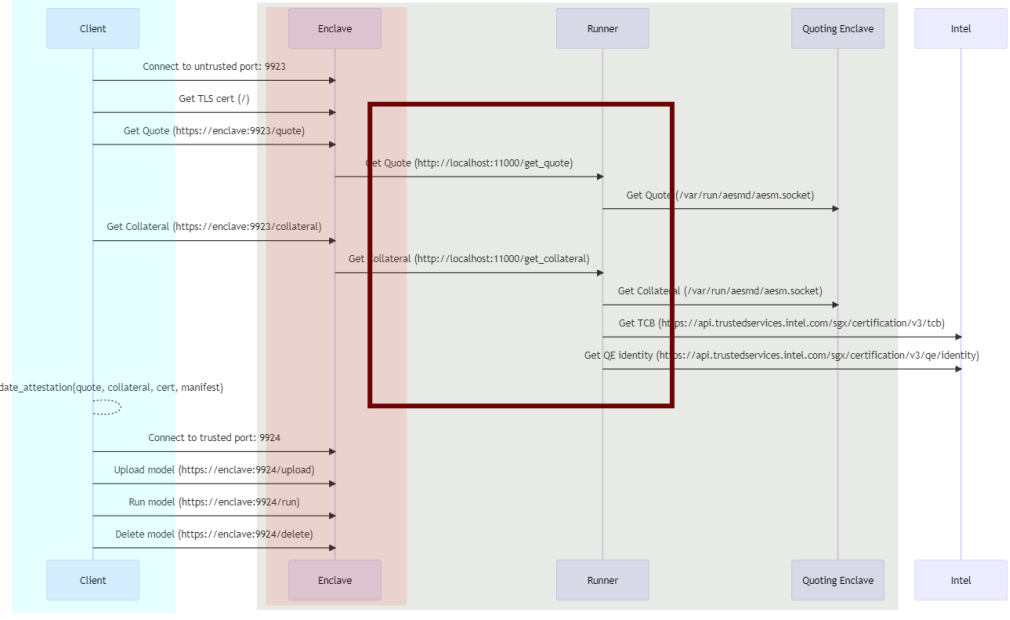
Cartography



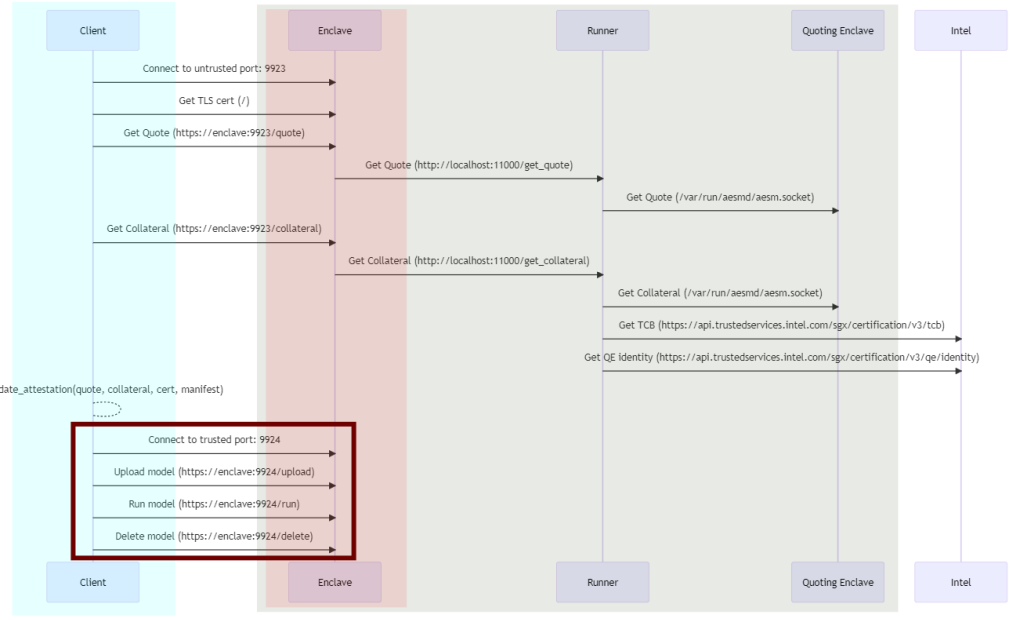
Threats


Client
Enclave
Runner
Quoting Enclave
Intel
Connect to untrusted port: 9923
Get TLS cert (/)
Get Quote (https://enclave:9923/quote)
Get Quote (http://localhost:11000/get_quote)
Get Quote (/var/run/aesmd/aesm.socket)
Get Collateral (https://enclave:9923/collateral)
Get Collateral (http://localhost:11000/get_collateral)
Get Collateral (/var/run/aesmd/aesm.socket)
Get TCB (https://api.trustedservices.intel.com/sgx/certification/v3/tcb)
Get QE identity (https://api.trustedservices.intel.com/sgx/certification/v3/qe/identity)
Call validate_attestation(quote, collateral, cert, manifest)
Connect to trusted port: 9924
Upload model (https://enclave:9924/upload)
Run model (https://enclave:9924/run)
Delete model (https://enclave:9924/delete)
Client
Enclave
Runner
Quoting Enclave
Intel

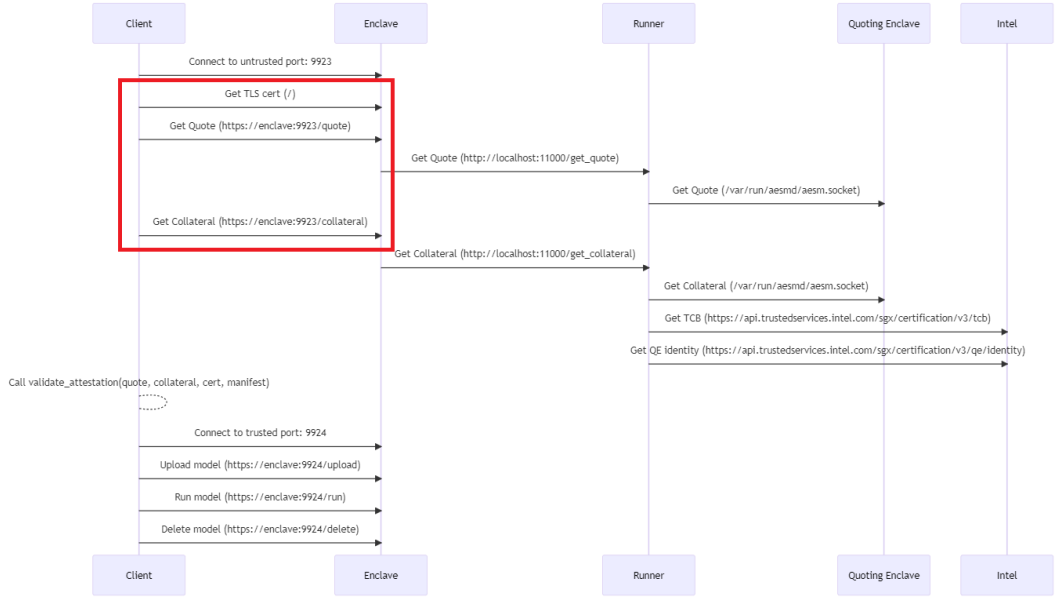
Threats


Client
Enclave
Runner
Quoting Enclave
Intel
Connect to untrusted port: 9923
Get TLS cert (/)
Get Quote (https://enclave:9923/quote)
Get Quote (http://localhost:11000/get_quote)
Get Quote (/var/run/aesmd/aesm.socket)
Get Collateral (https://enclave:9923/collateral)
Get Collateral (http://localhost:11000/get_collateral)
Get Collateral (/var/run/aesmd/aesm.socket)
Get TCB (https://api.trustedservices.intel.com/sgx/certification/v3/tcb)
Get QE identity (https://api.trustedservices.intel.com/sgx/certification/v3/qe/identity)
Call validate_attestation(quote, collateral, cert, manifest)
Connect to trusted port: 9924
Upload model (https://enclave:9924/upload)
Run model (https://enclave:9924/run)
Delete model (https://enclave:9924/delete)
Client
Enclave
Runner
Quoting Enclave
Intel

Threats


Client
Enclave
Runner
Quoting Enclave
Intel
Connect to untrusted port: 9923
Get TLS cert (/)
Get Quote (https://enclave:9923/quote)
Get Quote (http://localhost:11000/get_quote)
Get Quote (/var/run/aesmd/aesm.socket)
Get Collateral (https://enclave:9923/collateral)
Get Collateral (http://localhost:11000/get_collateral)
Get Collateral (/var/run/aesmd/aesm.socket)
Get TCB (https://api.trustedservices.intel.com/sgx/certification/v3/tcb)
Get QE identity (https://api.trustedservices.intel.com/sgx/certification/v3/qe/identity)
Call validate_attestation(quote, collateral, cert, manifest)
Connect to trusted port: 9924
Upload model (https://enclave:9924/upload)
Run model (https://enclave:9924/run)
Delete model (https://enclave:9924/delete)
Client
Enclave
Runner
Quoting Enclave
Intel

MITM

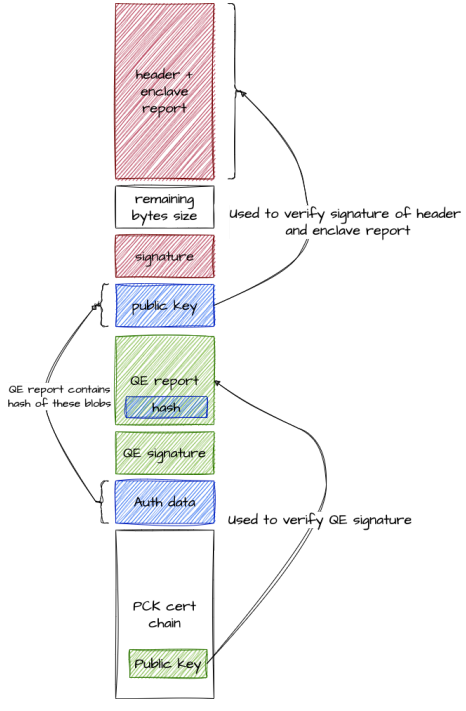

Client
Enclave
Runner
Quoting Enclave
Intel
Connect to untrusted port: 9923
Get TLS cert (/)
Get Quote (https://enclave:9923/quote)
Get Quote (http://localhost:11000/get_quote)
Get Quote (/var/run/aesmd/aesm.socket)
Get Collateral (https://enclave:9923/collateral)
Get Collateral (http://localhost:11000/get_collateral)
Get Collateral (/var/run/aesmd/aesm.socket)
Get TCB (https://api.trustedservices.intel.com/sgx/certification/v3/tcb)
Get QE identity (https://api.trustedservices.intel.com/sgx/certification/v3/qe/identity)
Call validate_attestation(quote, collateral, cert, manifest)
Connect to trusted port: 9924
Upload model (https://enclave:9924/upload)
Run model (https://enclave:9924/run)
Delete model (https://enclave:9924/delete)
Client
Enclave
Runner
Quoting Enclave
Intel

MITM

- We can modify the following inputs:
  - the public enclave certificate;
  - the quote;
  - the collateral.
- Our goal was to make the client trusts us either by bypassing its checks or by exploiting a vulnerability in the parsing.

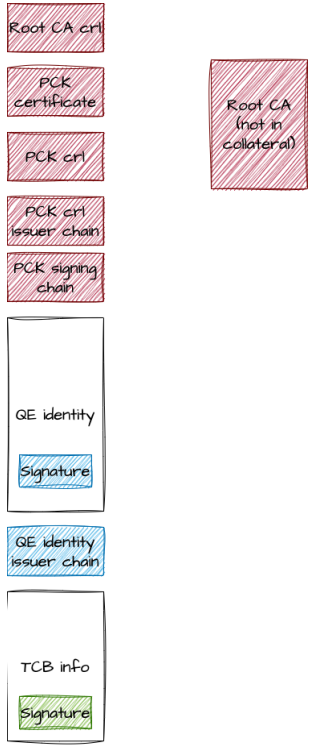
Quote

- REPORT
  - Hardware report generated by the Intel SGX HW that provides identity and measurement information of the enclave and the platform.
- Quoting Enclave (QE)
  - A special enclave on every SGX processor which is tasked entirely with handling the remote attestation.
  - It receives REPORTs from other enclaves, verifies them and signs them with the attestation key before returning the result, also known as a QUOTE, to the application.



Collateral

- Trusted Computing Base Info (TCB Info)
  - To check if hardware is up-to-date
- Provisioning Certification Key (PCK)
  - Signing key available to the PCE
  - The key is unique to the processor package or platform instance and its TCB
  - The public part of the key is distributed as a PCK certificate.



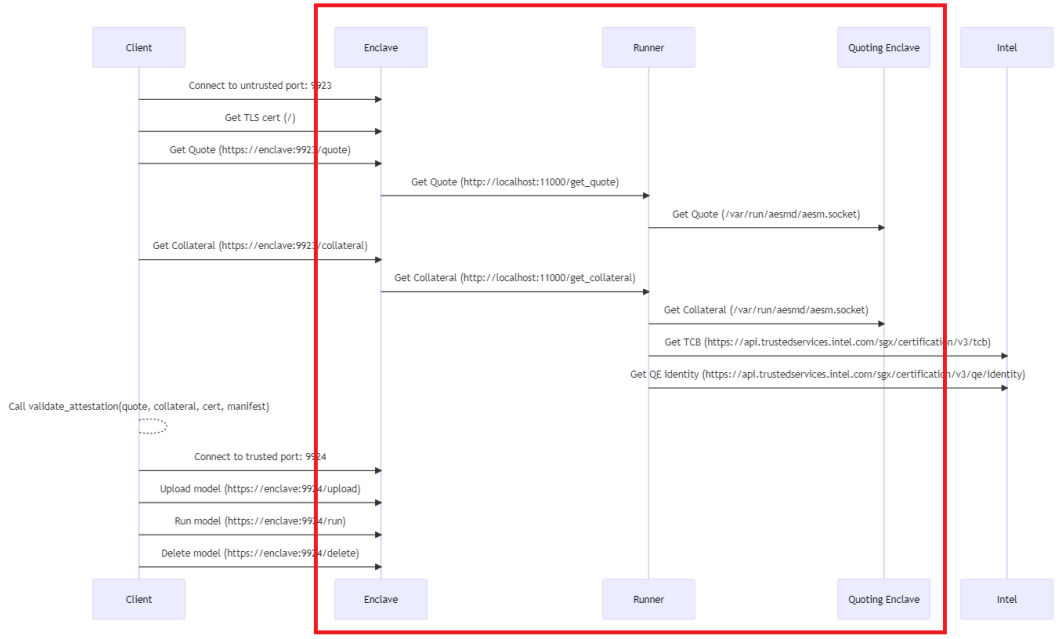
Client

- Even if the code of the client is not in the scope of this audit and the client is trustworthy
- We need to take a look on what could happen on the client side with the inputs controlled by the attacker
- Most of the work to check if the quote and the collateral are valid is done by the QVL (Quote Verification Library).
- This library is written in C++ by Intel.

Client

- To verify that the enclave is genuine, after fetching the certificate, the quote, and the collateral, the client does a lot of checks
  - We did not find a way to bypass all these checks
  - But since the client was not in the scope, we just verified that the checks were sound
- We didn't try to fuzz the parsing code (`openssl` and `rapidjson` are heavily used)
- Since the client has all the sensible data, a vulnerability in a library used to parse or validate these information can be catastrophic

IAGO


Client
Enclave
Runner
Quoting Enclave
Intel
Connect to untrusted port: 9923
Get TLS cert (/)
Get Quote (https://enclave:9923/quote)
Get Quote (http://localhost:11000/get_quote)
Get Quote (/var/run/aesmd/aesm.socket)
Get Collateral (https://enclave:9923/collateral)
Get Collateral (http://localhost:11000/get_collateral)
Get Collateral (/var/run/aesmd/aesm.socket)
Get TCB (https://api.trustedservices.intel.com/sgx/certification/v3/tcb)
Get QE identity (https://api.trustedservices.intel.com/sgx/certification/v3/qe/identity)
Call validate_attestation(quote, collateral, cert, manifest)
Connect to trusted port: 9924
Upload model (https://enclave:9924/upload)
Run model (https://enclave:9924/run)
Delete model (https://enclave:9924/delete)
Client
Enclave
Runner
Quoting Enclave
Intel

Intel SDK

- When you want to write an enclave in Rust, you have mostly 2 choices:
  - Use Teaclave
  - Use Fortanix EDP
- Teaclave is using bindings to Intel SDK libraries, which means that, under the hood, you are still using Intel SDK libraries.
- Hence, the design of your application is dictated by the usage of this SDK

```
/* Arrays.edl - Samples for array attributes. */
enclave{
    /*
    *  Only for fixed-size array (size is explicitly specified).
    */
    trusted{
        /*
        * []:  can be used to declare an array.
        * [user_check]:
        * pointer of the array won't be valified, and the buffer pointed by 'arr'
        * is not copied into the enclave either. Enclave can modify the memory outside.
        */

        public void ecall_array_user_check([user_check] int arr[4]);
        /*
        * [in]:
        * buffer for the array will be allocated inside the enclave,
        * content of the array will be copied into the new allocated memory inside.
        * Any changes performed inside the enclave will not affect the array outside.
        */
        public void ecall_array_in([in] int arr[4]);
        /*
        * [out]:
        * buffer for the array will be allocated inside the enclave,
        * but the content of the array won't be copied. After ECALL returns,
        * the buffer inside the enclave will copied into outside array.
        */
        public void ecall_array_out([out] int arr[4]);
        /*
        * [in, out]:
        * buffer for the array will be allocated inside the enclave,
        * the content of the array will be copied either. After ECALL returns,
        * the buffer inside the enclave will by copied into outside array again.
        */
        public void ecall_array_in_out([in, out] int arr[4]);
        /*
        * [isary]:
        * tells Edger8r the user defined 'array_t' is an array type, 'arr' will be
        * treated as a pointer, no memory copied either due to [user_check].
        * For OCALLs, 'arr' shall point to the memory outside the enclave.
        */
        public void ecall_array_isary([user_check, isary] array_t arr);
    };
    untrusted{
        /*
        * [user_check|in|out|in,out|isary] can also be used in OCALLs, refer to the
        * "User Guide" for details.
        */
    };
};
```

Intel SDK

- For example, you need to split your application into two:
  - the trusted part, also called the enclave and
  - the untrusted part, called the app.
- You also need to write an EDL to specify which ECALLs and OCALLs will be used, as well as other required features.
  - ECALL: A call from the application into an interface function within the enclave.
  - OCALL: A call made from within the enclave to the application.

```
/* Arrays.edl - Samples for array attributes. */
enclave{
    /*
    *  Only for fixed-size array (size is explicitly specified).
    */
    trusted{
        /*
        * []:  can be used to declare an array.
        * [user_check]:
        * pointer of the array won't be valified, and the buffer pointed by 'arr'
        * is not copied into the enclave either. Enclave can modify the memory outside.
        */

        public void ecall_array_user_check([user_check] int arr[4]);
        /*
        * [in]:
        * buffer for the array will be allocated inside the enclave,
        * content of the array will be copied into the new allocated memory inside.
        * Any changes performed inside the enclave will not affect the array outside.
        */
        public void ecall_array_in([in] int arr[4]);
        /*
        * [out]:
        * buffer for the array will be allocated inside the enclave,
        * but the content of the array won't be copied. After ECALL returns,
        * the buffer inside the enclave will copied into outside array.
        */
        public void ecall_array_out([out] int arr[4]);
        /*
        * [in, out]:
        * buffer for the array will be allocated inside the enclave,
        * the content of the array will be copied either. After ECALL returns,
        * the buffer inside the enclave will by copied into outside array again.
        */
        public void ecall_array_in_out([in, out] int arr[4]);
        /*
        * [isary]:
        * tells Edger8r the user defined 'array_t' is an array type, 'arr' will be
        * treated as a pointer, no memory copied either due to [user_check].
        * For OCALLs, 'arr' shall point to the memory outside the enclave.
        */
        public void ecall_array_isary([user_check, isary] array_t arr);
    };
    untrusted{
        /*
        * [user_check|in|out|in,out|isary] can also be used in OCALLs, refer to the
        * "User Guide" for details.
        */
    };
};
```

Fortanix

- On the contrary, Fortanix EDP (Enclave Development Platform) applications are just like native Rust applications.
- They have a main function, can have multiple threads, and can make network connections.
- You do not have to write any *untrusted* code that runs outside the enclave since all the features are provided for you.

Fortanix

- You just need to compile for the `x86_64-fortanix-unknown-sgx` target as Fortanix EDP is fully integrated with the Rust compiler.
- Outside the enclave, the `enclave-runner` crate takes care of loading the enclave.
- Once the enclave is loaded, it provides a shim layer between the usercalls coming from the enclave, and the system calls needed to talk to the outside world.

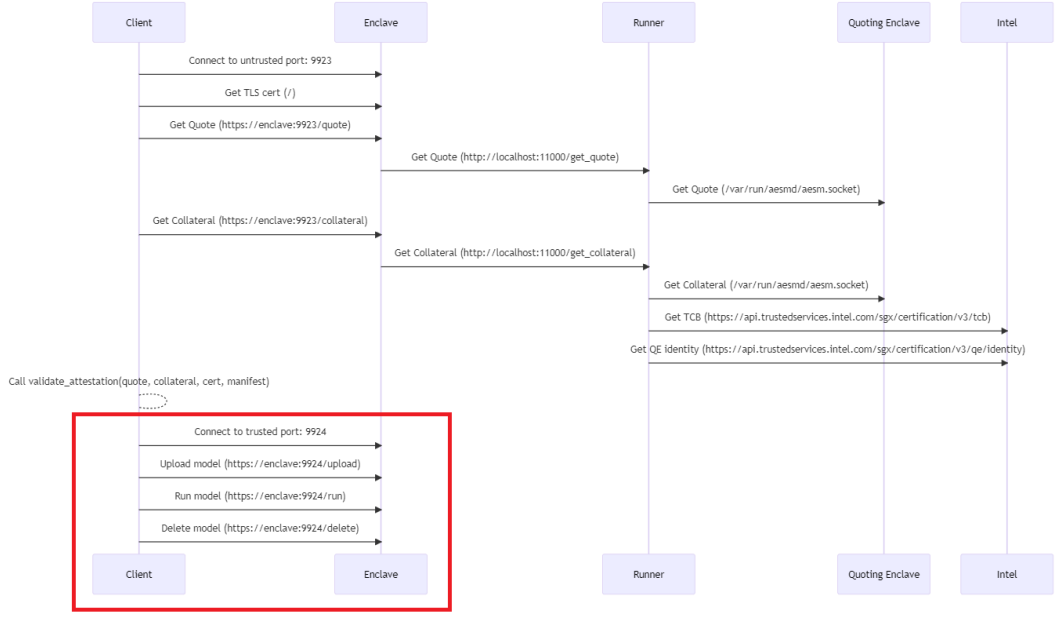
Fortanix

- Just like a normal OS, the Fortanix EDP defines an API and ABI that applications use to interact with the environment.
  - The usercall API has been specifically designed for use with SGX and consists of less than 20 different calls.
  - Common patterns such as buffer-passing are implemented consistently across all the calls.
  - The interface is kept small intentionally to aid security audits.
- In particular, the Fortanix interface has been designed to avoid known attacks on enclave interfaces, such as Iago attacks and information leakage through structure padding.
- In addition, the Rust type system is used to disallow direct access to user memory from the enclave and avoid time-of-check to time-of-use (TOCTTOU) issues.

Fortanix

- There are several differences with traditional enclaves
  - For example, it's not straightforward to see where ECALLS/OCALLS are made (no EDL)
  - EPC handling is masked because everything goes through the Rust allocator, and so on.
- Also, Fortanix EDP seems to only support SGX1
- It means that Fortanix EDP work well on a processor with SGX2 support but it will not use the new features (for example there is no support for dynamic memory).
- Fortanix was already audited against IAGO attacks and was found quite robust
- During the audit, we didn't have time to conduct Iago attacks

Malicious client



Malicious client

- We did not try to fuzz the models, as it didn't seem like a relevant thing to do
- The enclave is written in Rust and do not use any unsafe code so the probability of finding a vulnerability leading to remote code execution is low

Malicious client

- Since Fortanix EDP only uses SGX1, all memory needs to be preallocated before launching the enclave
- In `Cargo.toml` you specify how much stack and heap you need, if the enclave memory consumption is higher, the enclave panics and crashes
- Mithril BlindAI (during the time of the audit) didn't provide a way to offload to disk the models that were sent

```
[package.metadata.fortanix-sgx]
# stack size (in bytes) for each thread, the default stack size
# is 0x20000.
stack-size=0x400000
# heap size (in bytes), the default heap size is 0x2000000.
heap-size=0xFBA00000
# the default number of threads is equal to the number of available
# CPUs of the current system.
# Gotcha: Don't forget to count the main thread when counting number
# of threads.
threads=20
# SSA frame size (in pages) for each thread, the default SSA frame
# size is 1.
# You normally don't need to change the SSA frame size.
ssaframesize=1
# whether to enable EDP debugging features in the enclave, debugging
# is enabled by default.
debug=false
```

Malicious client

- The client uses a configuration file which contains sensible things
- If an attacker is able to modify it (for example by allowing a debugging enclave), it is quite simple to bypass the enclave

```
# Enclave manifest file
# Determines which enclaves are to be accepted by
# the client

# Enclave measurement
# MRENCLAVE represents the enclave's contents and build process
mr_enclave = "5ad7f8619b5d0d607653c1b292096fce14b75915af817e
              3195ea0cab47250cfe"

# Set to true to allow enclave running in DEBUG mode
# A production service should never allow debug-mode enclaves
allow_debug = true

# Enclave attributes are formed of :
#  * attributes_flags
#  * attributes_xfrm (XFRM for XSAVE Feature Request Mask)
# The allowed attributes are described by bitmasks.
# The layout of the structures are described in Intel documentation

# ATTTRIBUTE default : 0x4
# 0x4 == MODE64BIT
attributes_flags_hex = "0x4"
# ATTTRIBUTEMASK default : ~0x2 = 0xfffffffffffffffd
# Check everything match the selected attributes
# except for the DEBUG field
# The DEBUG field value is checked separately
# via the `allow_debug` parameter
attributes_mask_flags_hex = "0xfffffffffffffffd"
# ATTRIBUTES.XFRM default : 0x3

...
```

Conclusion

Conclusion

- We didn't find any flaws (except for the DOS if too many models are uploaded)
  - But putting the client out-of-scope excludes de-facto more simple ways of compromission
- Fortanix EDP allows to easily and quickly write enclaves
- Auditing Rust enclaves written with Fortanix is quite hard, specially if you want to conduct Iago attacks
- It's quite difficult to determine if the system is fully patched, since TCB info provided by the host are not accurate in a virtual machine environment
  - Need to bypass the PCS and request directly the collateral from Intel servers

Quarkslab

Questions?

Appendix

Trusted Computing Base (TCB)

The TCB, also called Enclave Measure, is a cryptographic stamp of all the build activities, which includes:

```
- Content: code, data, stack, and heap;
- Location of each page within the enclave;
- Security flags being used.
```

REPORT

- Hardware report generated by the Intel® SGX HW that provides identity and measurement information of the enclave and the platform.
- It can be MAC'd with a key available to another enclave on the same platform.

Quoting Enclave (QE)

- A special enclave on every SGX processor which is tasked entirely with handling the remote attestation.
- It receives REPORTs from other enclaves, verifies them and signs them with the attestation key before returning the result, also known as a QUOTE, to the application.

DCAP

- DCAP is the software infrastructure provided by Intel for the new ECDSA/PCS-based remote attestation.
- It relies on the Flexible Launch Control hardware feature and is intended for a server environment where you control all the machines.

AESM client

- AESM stands for Architectural Enclave Service Manager, which handles all system services for SGX enclaves such as the trusted time attestation and monodic counters with the help of the Platform Service Enclave PSE.

Provisioning Certification Enclave (PCE)

- Intel SGX architectural enclave that uses a PCK to sign QE REPORT structures for Provisioning or Quoting Enclaves
- These signed REPORTS contain the ReportData indicating that attestation keys or provisioning protocol messages are created on genuine hardware.

Provisioning Certification Key (PCK)

- Signing key available to the PCE
- The key is unique to the processor package or platform instance and its TCB
- The public part of the key is distributed as a PCK certificate.

SVN

- The version number that indicates when security-relevant updates occurred.
- New versions can have increased functional versions without incrementing the SVN.

FMSPC

- Description of the processor package or platform instance including its Family, Model, Stepping, Platform type, and customized SKU (Stock Keeping Unit).

Azure patch

```
if std::env::var("BLINDAI_AZURE_DCS3_PATCH").is_ok() {
    println!("The patch for Azure DCsv3 and DCdsv3-series VMs is enabled. Requesting collateral directly from Intel, bypassing the PCS.");

    let api_tcb_info_response = ureq::get("https://api.trustedservices.intel.com/sgx/certification/v3/tcb")
        .query("fmspc", &hex::encode(fmspc)).call()?;

    tcb_info_issuer_chain = urlencoding::decode(
        api_tcb_info_response
            .header("SGX-TCB-Info-Issuer-Chain")
            .unwrap(),
    )?
    .to_string();

    tcb_info = api_tcb_info_response.into_string()?;

    let api_qe_identity_response = ureq::get("https://api.trustedservices.intel.com/sgx/certification/v3/qe/identity").call()?;
    qe_identity_issuer_chain = urlencoding::decode(
        api_qe_identity_response
            .header("SGX-Enclave-Identity-Issuer-Chain")
            .unwrap(),
    )?
    .to_string();

    qe_identity = api_qe_identity_response.into_string()?;
}
```

Collateral TCB Info


```
{
  "signature": "b956dfd032434a1d638b11d007a947c5bf0573a320252b6589b9d985421cfb0a6d748a2b9d05c3e29b65c38e41ee2e17522ec92bfa13eb4009a7cf99b3831f4d",
  "tcbInfo": {
    "fmspc": "00606a000000",
    "issueDate": "2021-04-20T18:27:52Z",
    "nextUpdate": "2021-05-20T18:27:52Z",
    "pceId": "0000",
    "tcbEvaluationDataNumber": 10,
    "tcbLevels": [
      {
        "tcb": {
          "pcesvn": 10,
          "sgxtcbcomp01svn": 4,
          "sgxtcbcomp02svn": 4,
          "sgxtcbcomp03svn": 3,
          "sgxtcbcomp04svn": 3,
          "sgxtcbcomp05svn": 255,
          "sgxtcbcomp06svn": 255,
          "sgxtcbcomp07svn": 0,
          "sgxtcbcomp08svn": 0,
          "sgxtcbcomp09svn": 0,
          "sgxtcbcomp10svn": 0,
          "sgxtcbcomp11svn": 0,
          "sgxtcbcomp12svn": 0,
          "sgxtcbcomp13svn": 0,
          "sgxtcbcomp14svn": 0,
          "sgxtcbcomp15svn": 0,
          "sgxtcbcomp16svn": 0
        },
        "tcbDate": "2020-11-11T00:00:00Z",
        "tcbStatus": "UpToDate"
      },
      ...
    ]
  }
  ...
}
```