Mithril BlindAI

Security assessment report

Reference 23-03-1142-LIV
Version 1.1
Date 2023/26/05

Quarkslab SAS
10 boulevard Haussman
75009 Paris
France

Contents

1 Project Information

Document history			
Version	**Date**	**Details**	**Authors**
1.0	2023/26/05	Initial Version	Ramtine Tofighi Shirazi & Damien Aumaitre & Dahmun Goudarzi
1.1	2023/26/05	Minor revisions	Ramtine Tofighi Shirazi & Damien Aumaitre & Dahmun Goudarzi

Quarkslab		
Contact	**Role**	**Contact Address**
Frédéric Raynal	CEO	`fraynal@quarkslab.com`
Ramtine Tofighi Shirazi	Project Manager	`mrtofighishirazi@quarkslab.com`
Damien Aumaitre	R&D Engineer	`daumaitre@quarkslab.com`
Dahmun Goudarzi	R&D Engineer	`dgoudarzi@quarkslab.com`

Mithril Security		
Contact	**Role**	**Contact Address**
Daniel Huynh	CEO	`daniel.huynh@mithrilsecurity.io`
Corentin Lauverjat	Head of Security Engineering	`corentin.lauverjat@mithrilsecurity.io`
Yassine Bargach	Security Engineer	`yassine.bargach@mithrilsecurity.io`

2 Executive summary

This report presents the results of the collaboration between Mithril Security and Quarkslab on the security audit of the BlindAI-preview[1].

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.

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.

This audit report starts with introductory sections and in Section 3 a more detailed presentation of the context and scope of the audit. Then, Section 4 introduces the discovery work performed by Quarkslab auditors to define the threat model as described in Section 5. Afterward, Section 6 presents Quarkslab auditors' cartography and assessment methodology as applied on the BlindAI. The results of the assessment are provided, based on the threat model in Section 7. Finally, Section 8 provides a conclusion based on the assessment performed on the security of the BlindAI, with respect to the above-mentioned scope of work.

The security audit was performed from January to March 2023.

[1] `https://github.com/mithril-security/blindai-preview`

2.1 Disclaimer

This report reflects the work and results obtained within the duration of the audit defined for 40 days on the specified scope and as agreed between Mithril and Quarkslab in *22-06-1001-PRO*. Tests are not guaranteed to be exhaustive and the report does not ensure the code is bug or vulnerability free.

Warning

The scope of the assessment was modified to the **BlindAI-preview** instead of the BlindAI at the beginning of the audit. The BlindAI-preview is based on other technical modules, such as Fortanix [1] (as opposed to Teaclave [2] for the BlindAI). Attacks on the client of the BlindAI-preview are only considered during *Man-in-the-Middle* attacks, considering an attacker on the server side. Other scenarios were considered out of scope of this assessment.

Throughout the rest of this report, **BlindAI** will be use to identify the audited **BlindAI-preview**.

2.2 Findings summary

The following table synthesizes the various findings that were uncovered during the audit. The severity classification given as informative, low, and medium, reflects a relative hierarchy between the various findings of this report. It depends on the threat model and security properties considered.

During the time frame of this assessment, the auditors identified:

- 1 vulnerability rated as low;
- 7 security weaknesses considered as informative only.

ID	Description	Category	Severity
LOW 1	BlindAI does not have a mechanism to offload to the disk when memory consumption is high, which makes a denial-of-service attack possible when multiple models are sent.	CWE-770: Allocation of Resources Without Limits or Throttling	Low
INFO 1	Enclaves are unsigned prior to running which is not compliant with Fortanix EDP documentation.	CWE-345: Insufficient Verification of Data Authenticity	Info
INFO 2	One of the dependencies used by the enclave, `rouille` [3] seems to have some issues.	CWE-1104: Use of Unmaintained Third Party Components	Info
INFO 3	The code used in `BlindAI` does not properly handles errors or exceptional conditions.	CWE-703: Improper Check or Handling of Exceptional Conditions	Info
INFO 4	Some functions are using `unwrap` (which can cause a panic) when they are supposed to handle errors.	CWE-676: Use of Potentially Dangerous Function	Info
INFO 5	When the runner requests a quote from the quoting enclave, it uses a nonce value set to 0 which is not a good practice.	CWE-323: Reusing a Nonce, Key Pair in Encryption	Info
INFO 6	Client uses `rapidjson` from Tencent which is known for not taking security issues [4].	CWE-1357: Reliance on Insufficiently Trustworthy Component	Info
INFO 7	Usage of CBOR to encode data sent and received by the enclave which not maintained anymore [5].	CWE-1104: Use of Unmaintained Third Party Components	Info

3 Context and scope

This chapter presents the context of the assessment, namely, BlindAI and Intel SGX. In order to get familiar with the context, the auditors present some key concepts and definitions.

Then, the audit settings and scope, with a client and a server, are also introduced.

3.1 BlindAI

BlindAI is an open-source solution offered by Mithril Security for confidential computing of AI inference on servers. It is based on a Rust server deployed on Azure VMs that have Intel SGX hardware. Thanks to this specific set of instructions provided by Intel, BlindAI can provide a space, called `enclave`, where the AI inference can be made securely and privately.

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 entire client-server protocol of BlindAI can be seen as follows:

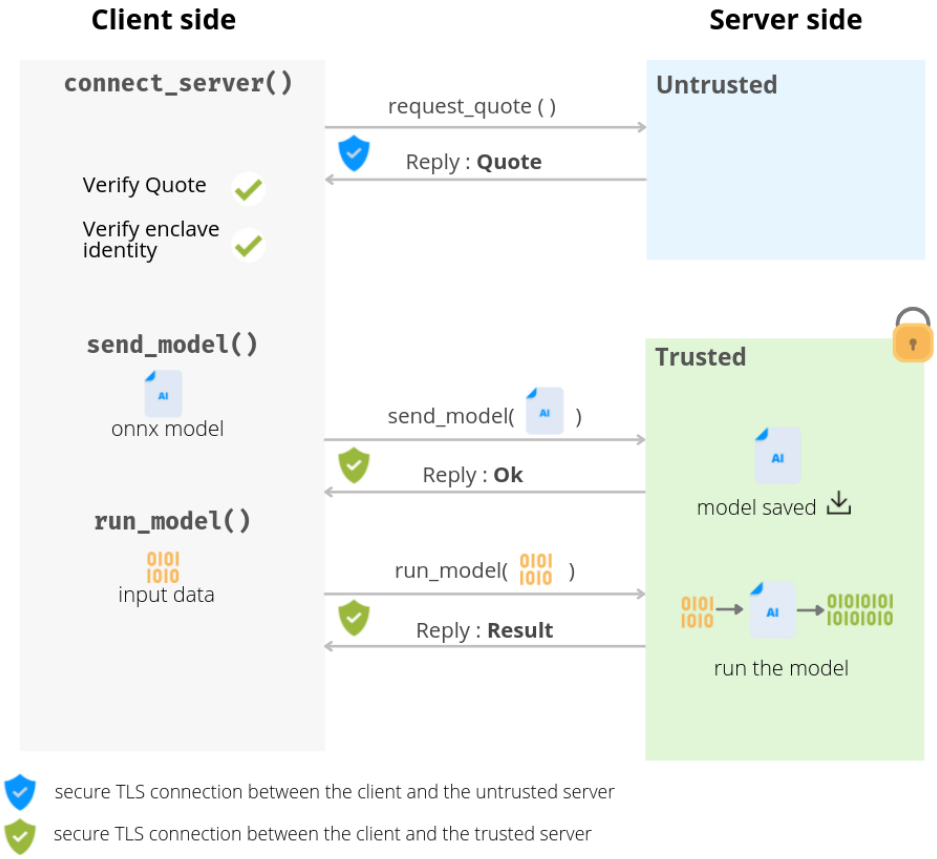

Figure 3.1: BlindAI Overview

Please note that this overview is taken from a previous version of BlindAI[1], but the overall design rationale is still relevant.

BlindAI has the advantage that it is an open-source project available on GitHub, and can be deployed on someone's infrastructure. The BlindAI preview is currently in the beta stage of development.

[1]Overview from `https://blindai-preview.mithrilsecurity.io/en/latest`

3.2 SGX

This section provides an introduction to the Intel SGX technology, along with some key concepts and definitions related to the product and used by the auditors to get acquainted with the BlindAI.

3.2.1 Overview

Intel SGX is a technology that was developed to meet the needs of the Trusted Computing industry, similar to the ARM TrustZone, but this time for desktop and server platforms. It allows user-land code to create private memory regions, called enclaves, that are isolated from other processes running at the same or higher privilege levels. The code running inside an enclave is effectively isolated from other applications, the operating system, the hypervisor, et cetera.

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.

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;
- multi-threading is supported (but not trivial to implement properly);
- an enclave can access its application's memory, but not the other way around.

`SGXv2` introduces a new set of instructions and hardware features that enables developers to dynamically manage memory within the SGX environment. It also allows us to extend the sizes of the pages (from 128 Mo to 16 Gb) to handle bigger data.

3.2.2 Key concepts and definitions

The paragraphs below introduce some key definitions and concepts used by the auditors for the discovery of the target of evaluation[2].

Attestation

Sometimes enclaves need to collaborate with other enclaves on the same platform due to different reasons such as data exchange if the enclave is too small to hold all the information, or communication with Intel reserved enclaves to conduct specific Intel services.

[2]Please note that most or some definitions are taken directly from [6], [7], and Intel's documentations.

Therefore, the two exchanging enclaves have to prove to each other that they can be trusted. In other scenarios when an SGX enabled ISV client requests secrets from its ISV client, password management service, for example, the client has to prove to the server that the client application is running on a trusted platform that can process the secrets securely. Both of those two conditions require a proof of secured execution environment, and Intel SGX refers to this proving process as *attestation*.

There are two types of attestation concerning the two above-mentioned scenarios:

1. Local Attestation;
2. Remote Attestation.

Sealing

In order to protect and preserve the data outside an enclave, a mechanism is in place which allows enclave software to retrieve a key unique to that enclave. This key can only be generated by that enclave on that particular platform. Enclave software uses that key to encrypt data to the platform or to decrypt data already on the platform. The terms *sealing* and *unsealing* are used by SGX to describe the process of encrypting and decrypting, respectively.

Trusted Computing Base (TCB)

The TBC, 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).

3.3 AI Inference

AI inference is the core component of machine learning algorithm and their operations.

In order to perform such complex operations, machine learning algorithms rely on so-called tensors, which are vectors and matrices in multi-dimensions containing several sources of information used as input or by the model to obtain desired results. They are key building blocks in the AI inference component and their privacy are of at most importance.

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.

BlindAI uses the ONNX format (see [8]) in order to manipulate tensors and models within their infrastructure. ONNX is an open framework for AI inference, widely used in the community and supported by most of the main actors in machine learning. It has the advantage of being interoperable and having built-in resources for hardware optimizations.

3.4 Audit Scope

As previously mentioned, 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 scope of work was define and agreed 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.

3.4.1 Virtual Machine

The table below presents the virtual machine details used during the security assessment.

Setup	Virtual machine provided by Mithril Security
Host	Azure VM DCs4v3
OS	Ubuntu 20.04
CPU	Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz

3.4.2 Server

The table below provides the version and commit hash used at the time of the audit:

Intel SGX SDK	version 2.11
Azure DCAP client	`39e0381b7ab7c672b92132f71583a5d565e6b16e`

3.4.3 Client

For the client side, the evaluation was made using the pip package release version 0.0.2 from January 25, 2023. The client also uses an external library developed by Intel for quote verification, on which a patch was applied, called QvL (Quote Verification Library). The patch can be found on `https://github.com/mithril-security/sgx-dcap-quote-verify-python`.

Client (pip package)	release version 0.0.2 (Jan. 25, 2023)
QvL (patched)	patch repository

In this audit, we assume that the client is used as it is and no modification to its setup can be made (for instance to its `toml` files).

Warning

Quarkslab auditors note that the above-mentioned hypothesis is made given the early development stages of BlindAI. This hypothesis makes some attack scenarios from the wild not applicable. However, this assumption was done to restrict the scope of the audit to scale to the given time frame.

4 Discovery

4.1 State of the Art on Intel SGX

In this section, we describe the different documentations and tools that have been studied by the auditors to build some knowledge bases on Intel SGX, from its inception to the most recent known attacks. We first start by listing different sources for understanding better how Intel SGX works, then give a list of the different known attacks and if possible link to their source codes. Finally, we describe the two main tools that are currently available to play with Intel SGX, either for side-channel-based attacks (SGX-Step) or fuzzing (SGX Fuzz).

4.1.1 Background on Intel SGX

Intel Software Guard Extension (SGX) [9] 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. A set of talks and tutorials were given by Intel to describe their new innovation [10, 11]. Due to the problem that SGX aims to solve, namely secure computing, this new technology gained interest in the community and started to receive a lot of scrutiny [12, 13].

One of the core articles about Intel SGX is made by two MITs researcher [14]. They compiled in a single document every valuable information provided by Intel's documentation and diverse research papers into one dense paper that explains in detail how exactly Intel SGX works, from hardware wires to high-level cryptographic protocols. Since 2016 and its publication, it has served as a ground basis for many papers or publications involving Intel SGX research. The paper first starts by describing the security problem Intel SGX tries to tackle: secure remote computation via a trusted computing device using software attestation. Then, the author provides a very rich and well-written background on computer architecture and security. Those backgrounds aim to cover all principles behind Intel's most popular computer processors, cryptographic primitives and protocols, software attestation, and known attacks in such context (either at the software or hardware level). After detailing some other trusted hardware devices such as the ARM TrustZone or previous attempts from Intel, a deep dive into the Intel SGX programming model and its analysis is made by the authors. In particular, they provide a very thorough explanation of the enclaves, the central concept behind SGX.

Following this work, a series of other publications exist in the wild providing a more succinct overview of Intel SGX. Amongst them, we can find the Quarkslab blog posts [7] or the SGX 101 website [6]. For enclaves in particular, SSLab made an interesting summary of their functioning [15].

Intel soon updated their micro-code to SGX2 [16], which introduces a new set of instructions and hardware features that enables developers to dynamically manage memory within the SGX

environment. It allows also to extend the sizes of the enclaves to handle bigger data. Intel SGX2 was introduced with Gemini Lake processors, but it is also supported on multiple other processors such as the Xeon family.

To gain the trust between a remote provider or producer and the hardware entity, Intel relies on remote attestation. On the one hand, remote attestation ensures to the client that the software is running inside an SGX enclave and on a fully updated system. On the other hand, the attestation guarantees to the hardware of the identity of the software being attested, the execution mode, and if the software was tampered or not. To provide remote attestation, Intel developed two main attestation methods: Intel EPID and Intel SGX DCAP.

Intel Enhanced Privacy ID (Intel EPID) [17] is the Intel security technology to handle anonymity and membership revocation. It is now deprecated for server environments, which means deprecated for Intel SGX (as Intel SGX is only approved for server/cloud usage).

Intel SGX Data Center Attestation Primitive (Intel SGX DCAP) [18] is the software infrastructure provided by Intel for the new ECDSA/PCS-based remote attestation. It relies on Flexible Launch Control hardware feature. It is intended for a server environment where you control all the machines.

More details on how the remote attestation work are given in Section 6.

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.

In recent years, developers have started implementations on Intel SGX targets for cloud usage in order to tackle different problems answered by secure computing. A good starting point to understand how to develop on SGX target and the different problematics inherent to SGX programming is INRIA's lesson on SGX basics [19]. It provides a good overview on how SGX and its different component works at the software level, the basics on how to configure an enclave and define the `ECALLs` and `OCALLs`, and some exercises on running sensitive data inside an enclave from scratch. A more industrial project on SGX development is the Gramine project, which offers a dedicated OS for cloud deployment to run applications in an isolated environment such as SGX [20]. The project is open source and written mostly in C. More recently, several developments have been provided by the Rust community to propose memory-safe enclave programming [21]. The most notable Rust project is Fortanix [22]. Fortanix-EDP is the main building block of the BlindAI and we provide in the following a more in-depth description and analysis (see Section 4.3).

In the following, we give an overview of most of the attacks published on Intel SGX in the last couple of years. Some notable papers that list several of those attacks, mostly the ones based on side-channel are the SGX Fail paper [23] and a survey made by a couple of academics on ArXiv [24]. To the best of our knowledge, most of the attacks have been thwarted by Intel via security patches and countermeasures.

4.1.2 List of known attacks on Intel SGX

There exists a certain number of attacks on Intel SGX since its release in 2015. Most of them are based on side-channel attacks, where an attacker can extract or recover viable information via controlled channel attacks, cache attacks, speculative execution attacks, branch prediction attacks, rogue data cache loads, micro-architectural data sampling and software-based fault injection attacks. We summarized a list of known attacks we could find from various sources. Please note that we might have not been fully exclusive on them and the table was made during the allocated time frame.

Publication	Summary	Source	Year
Controlled channel attacks			
Controlled-Channel [25]	Memory access monitoring via page fault	N/A	2015
Stealthy Page Table-Based [26]	Observe enclave page accesses by exploiting the address translation process.	[27]	2017
SGX-Step [28]	Configure APIC timers and tracks page table entries directly from user space	[29]	2017
Nemesis [30]	Leak micro-architectural instruction timings from enclaved execution environments	[31]	2018
Off-Limits [32]	Exploitation of the memory segmentation feature in 32-bit legacy mode	N/A	2018
Leaky Cauldron [33]	Exploiting risk in memory management, from TLB to DRAM modules	[34]	2017
CopyCat [35]	Controlled-channel attack that deterministically counts the number of instructions executed within a single enclave code page	N/A	2020
Cache attacks			
CacheZoom [36]	Cache side-channel attack to virtually track all memory accesses of SGX enclaves	[37]	2017
Cache Attacks on Intel SGX [38]	Access-driven cache-timing attack on AES when running in an Intel SGX enclave	N/A	2017
Malware Guard Extension [39]	Prime+Probe cache attack on a co-located SGX enclave	N/A	2017
Software Grand Exposure [40]	Cache attack on SGX enclave that is claimed easier to deploy and less detectable	N/A	2017
CacheQuote [41]	Implementation weakness of the EPID protocol that leaks information via cache side-channel	N/A	2018
MemJam [42]	Exploits false dependency of memory read-after-write	N/A	2017
Rogue Data Cache Loads			
FORESHADOW [43]	Software micro-architectural attack to dismantle SGX's security objectives	[44]	2018

Figure 4.1: List of known attacks on SGX - Part 1

Publication	Summary	Source	Year
Speculative Execution Attacks			
SgxPectre [45]	SGX-variants of Spectre attacks	[46]	2019
Spectre Returns [47]	Spectre-like attack that exploits return stack buffer instead of branch predictor unit	N/A	2018
SGXSpectre	Sample code demonstrating a Spectre-like attack against an Intel SGX enclave	[48]	2018
Branch Prediction Attacks			
Fine-grained Control Flow [49]	Reveals fine-grained control flows in an enclave with branch shadowing	N/A	2017
BranchCode [50]	Infer the direction of an arbitrary conditional branch instruction	N/A	2018
Bluethunder [51]	Bypass existing protection and reveal the secret information inside an enclave with a new pattern history table based side-channel attack	N/A	2019
Micro-architectural Data Sampling			
RIDL [52]	Leak arbitrary data across address spaces and privilege boundaries via speculative unprivileged and constrained attacks	[53]	2019
ZombieLoad [54]	Meltdown-type attack that faults load instructions	[55]	2019
CacheOut [56]	Speculative execution attacks to leak data from OS kernel	[57]	2021
SGXAxe [58]	Follow-up of CacheOut	[57]	2021
CROSSTALK [59]	Combination of micro-coded instructions for off-core requests and MDS to reveal internal CPU state	[60]	2021
Software-based Fault Injection Attacks			
Plundervolt [61]	Exploitation of an undocumented Intel Core voltage scaling interface to corrupt integrity of SGX enclave computations	[62]	2020
VOLTpwn [63]	Software-based attack that affects the integrity of computation on x86 platform by targeting a physical core to force non-recoverable hardware faults	[64]	2020

Figure 4.2: List of known attacks on SGX - Part 2

Publication	Summary	Source	Year
	Software-based attacks		
AsyncShock [65]	Hijack enclave control flow or bypass access control by exploiting use-after-free and time-of-check-to-time-of-use	N/A	2016
Game of threads [66]	Schedule asynchronous threads to bypass enclave protections	[67]	2020
SmashEx [68]	Exploitation of asynchronous exceptions in SGX to expose enclave private memory and ROP attacks, without memory errors, side channels, or logic flaws	[69]	2021
A Tale of Two Worlds [70]	Sanitization vulnerabilities at the ABI and API level to exploit memory safety and side-channel attacks in the compiled enclave	[71]	2019
The Guard's Dilemma [72]	Two code-reuse attacks against enclave that provide full control of the CPU's general-purpose register without kernel privileges	N/A	2018
MicroScope [73]	Micro-architectural replay attack by inducing pipeline flushes which can denoise the port contention channel of execution units	[74]	2020
Platypus [75]	Software-based power side-channel attacks to exploit unprivileged access to the Intel Running Average Power Limit interface	[76]	2021

Figure 4.3: List of known attacks on SGX - Part 3

We described hereafter some of the most notable attacks.

RDRAND [77]

RDRAND is an Intel instruction that allows to grant access to a FIPS-validated random number generator. Most of the Intel SGX platform uses it as a source of fresh randomness. In mid-2020, a CVE was published where the authors were able to get an output of the RDRAND instructions due to some shortcut taken by Intel inside the core of the chip regarding speculative execution. This CVE has a tremendous impact since leaking the output of the RDRAND instruction can directly lead to the knowledge of the session's encryption keys.

Intel has since then released a series of microcode updates for the affected platforms.

LVI Attack [78]

LVI (Load Value injection) is an attack published at S&P 2020 [79] based on speculative execution to leak secret data. The attack is based on a micro-architectural flaw to inject malicious data through hidden processor buffers, by inducing a faulting load in the victim program and then recovering secret-dependent traces via side channels on the load value that has been transiently injected into code gadgets. This attack uses the SGX-Step framework. A demo is available on the webpage [78].

Intel has since then released microcodes updates for the affected platforms and future hardware products can mitigate this attack with silicon fixes.

SGXAxe [57]

CacheOut and its evolution SGXAxe are attacks published at S&P 2021 [56] based on speculative execution attacks to leak data from OS kernel, co-resident virtual machines, and SGX enclaves even when side-channel countermeasures are applied. Contrary to the previous attack, it has the advantage of selecting which data to leak instead of waiting for the data to be available. To do so, the authors propose a micro-architectural attack that bypasses Intel's buffer overwrite countermeasures by observing that when data is evicted from the CPU L1's cache, it is often transferred back to the leaky CPU buffer. A demo is available on the webpage [57].

Intel has since then released micro-code updates and BIOS updates to mitigate the attack.

AEPIC Leak [80]

AEPIC Leak is an attack published at USENIX 2022 [81] targeting most 10th, 11th and 12th generation Intel CPUs. The attack is based on an architectural bug where the attacker managed to recover sensitive data that has been directly in the cache due to an incorrectly defined range in the APIC MMIO. In order to mount the attack, one requires to be an administrator or root to access the APIC MMIO. This attack uses the SGX-Step framework. A demo is available on the webpage [80].

Intel has since then released microcode and SGX SDK updates to mitigate this attack.

4.1.3 List of found tools for Intel SGX studies

In this section, the auditors present some of the most notable tools related to Intel SGX state-of-the-art presented before.

SGX-Step

SGX-Step [82] is an open-source framework that allows to perform side channel attacks on Intel SGX platforms. It is based on the development of a malicious tool that allows to deploy a kernel module that will help researchers gather precious information on a targeted enclave via different entry points. It has been the entry point for several attacks and their GitHub repository lists and gives access to most of them: Plundervolt, AEPIC leak, LVI, a tale of two worlds, and so on.

The framework can be found on GitHub [29]. Their README details a certain number of known projects that uses SGX-Step. Please note that all Intel CPUs are supported for the framework [83].

SGX Fuzzer

At USENIX 2022, researchers presented a tool to perform fuzzing on Intel SGX platforms to expose enclave vulnerabilities even without source-code access [84]. With their open-source tool SGXFuzz [85], they were able to obtain several results on Intel SGX enclaves:

- Enclave dumping: automatically extracts executable enclave code;
- Enclave runner coverage: execution of enclave code outside SGX to retrieve code coverage feedback;
- Memory Location Havoc: analysis of the enclave memory to detect trust-boundary attacks;
- Fuzz analysis: security analysis of several enclaves which lead to 79 bugs or vulnerabilities found.

In order to make the tool work, an Intel PT-enabled CPU is required for the fuzzing setup. We discovered this tool during the state-of-the-art write-up and did not have access to such equipment in the time frame of this audit.

4.2 Discovery

4.2.1 Build

In this section, we describe the different steps we followed in order to build and run the BlindAI server and client. For the server, we followed the instructions provided in [86]. For the client, we followed the instructions provided in [87] (pip install) and [88] (client SDK installation).

Server

In order to build BlindAI on the server side, one simply needs to run the following commands:

```
$ just build
```

In order to understand what the build instruction does, we took a closer look at the `just` file. Here are the different steps performed in order to produce a functioning server build. Please note that some SGX-specific terms are employed, which are detailed in more depth in the next sections of the report. We focus here on the build process.

The `just` file first starts by building the Rust project for the SGX target:

```
$ cargo build -target x86_64-fortanix-unknown-sgx --release
```

Then, the initialization of the enclave is made via an SGX stream format command. This allows us to get a proper description of the basics of the enclave, and combined with a signature and initialization token, to produce the `MRENCLAVE`. In fact, one simply needs to compute the hash `SHA-256` of the output of the format command done in the `just` file:

```
$ ftxsgx-elf2sgxs "$binpath" \
  --heap-size 0xFBA00000 \
  --ssaframesize 1 \
  --stack-size 0x20000 \
  --threads 20
```

Then, the manifest file is generated by applying a hash on the output of the previous command which is saved into a TOML file as follows:

```
just generate-manifest-dev "$binpath.sgxs"
cp manifest.dev.toml client/blindai_preview/manifest.toml

just generate-manifest-prod "$binpath.sgxs"
```

Finally, the server's runner is built as follows:

```
$ ( cd runner && cargo build --release )
```

Client

BlindAI's client only requires `Python 3.8` or greater, `wget` and `pip` for its installation. Then, one can simply run the following command:

```
$ pip install blindai-preview
```

For the client SDK, it can be installed from the source code project simply as follows:

```
$ cd client
$ poetry install
$ poetry shell
```

4.2.2 Run

Server

The server can be started as follows:

```
$ just run
```

Then to run the enclave, one just needs to launch the runner with the following command:

```
$ ./runner/target/release/runner "$binpath.sgxs"
```

Client

The client can operate three operations on models with the server for AI inference: `upload`, `run`, and `delete`. This can simply be done in Python using the following template:

```
import blindai_preview

# Server connection
client = blindai_preview.connect(addr="YOUR_SERV_ADRESS", simulation_mode=True,
↪  hazmat_http_on_untrusted_port=True)

# Upload model to server
response = client_1.upload_model(model="./YOUR-MODEL.onnx")
MODEL_ID = response.model_id

# Data owner runs model by providing model_id and uploading data
pos_ret = client.run_model(MODEL_ID, data)

# AI company deletes model after use
client_1.delete_model(MODEL_ID)
```

```
# Disconnect from sever
client.close()
```

4.2.3 Code Structure

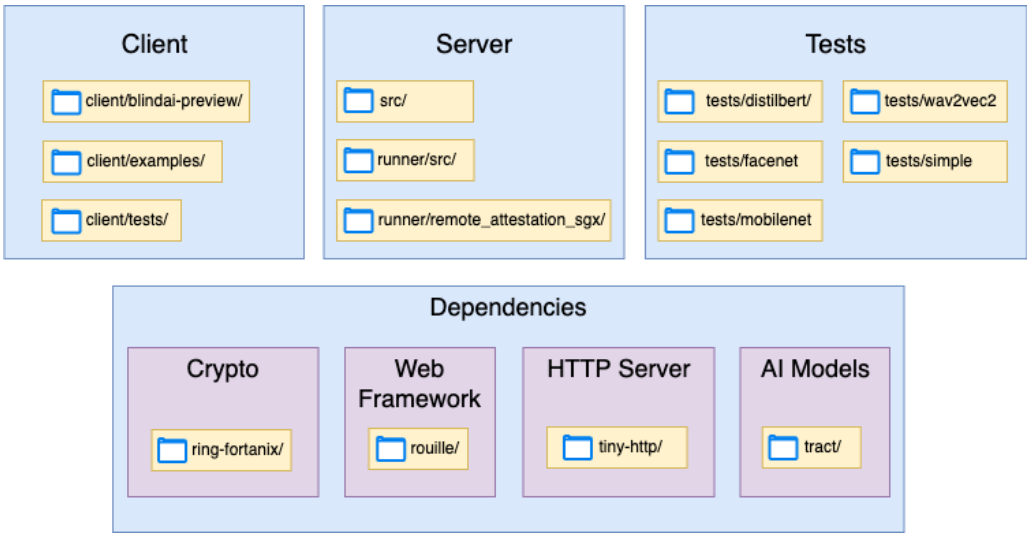

Figure 4.4: Code Structure Overview

The BlindAI-preview source code is composed of three main components:

1. a client;
2. a server;
3. and integration tests.

It is also composed of different dependencies such as a crate for cryptographic operations or model inference. The overall code structure is illustrated in Figure 4.4.

4.2.4 Client

The client source code is a Python based implementation. It is composed of three subfolders: `blindai-preview`, `examples`, and `test`.

In `blindai-preview`, the tensor and model classes are defined, on top of their respective methods for manipulations, computations, and serializations. It is also composed of functions for server connection, both on the untrusted and trusted ports. Finally, the three actions that a client can perform on a model are implemented: upload, run, and delete. The folder also contains the Intel provisioning root CA certificate for attestation.

In `examples` and `tests`, you can find some unfinished code most likely used for testing purposes and verification during the ongoing development.

4.2.5 Server

The server source is a Rust-based implementation. It is composed of two main subfolders: `src` and `runner`.

Source

In the `src` files, the main functions to perform remote attestation are implemented. Based on external dependencies, the server's code also implements functions for client requests handling and secure computation evaluation. Threads are defined to listen to clients' requests on the trusted port and schedule them. Functions for model inference (and storage) on ONNX-based model are developed on enclaves. There is also a function for the generation of the TLS certificate for secure SSL establishment with a client.

Runner

The `runner` folder is composed of two subfolders: a source one to implement a function for enclave initialization and running, and subfolder for generation and verification of quotes for remote attestation on Intel SGX.

4.2.6 Tests

Overall test of the blindai-preview features, to connect, upload, run, and delete the different models and then check the correctness of the inference results made via secure computing on SGX.

4.2.7 Dependencies

BlindAI is based on several external crates, listed hereafter.

Ring

Ring is a well-established Rust crate for cryptographic operations.

Rouille

`rouille` is a Rust crate allowing to create listening sockets and handling HTTP requests from the client.

tiny-http

`tiny-http` is a Rust crate to build HTTP server. To provide HTTPS, the crate relies either on OpenSSL or Rustls.

Tract

Tract is a well-known crate of ONXX format-based model inference.

Warning

We would like to point out that some dependencies used in BlindAI are not the most robust ones. In fact, `rouille` and `tiny-http` are crates that do not have proper maturity and scrutiny, or are not properly maintained. The audits of the external dependencies are out of the scope of this report.

- `rouille`: to the best of our knowledge there is no proper audit of this crate. Moreover, it seems to be maintained by a singular individual working on the crate once or twice per year for new releases.
- `tiny-http`: to the best of our knowledge there is no proper audit of this crate. Moreover, the latest version had failing CI since January 2019.
- `cbor`: there is no more maintenance on the `cbor` crate. However, the auditors note that Mithril Security provides internal supports to `cbor` and uses their own fork of the project.

Please note that all of those external dependencies (except ring and tract) were not required in BlindAI.

4.3 Fortanix

One of the main differences between `BlindAI` and `BlindAI preview` is the switch from `teaclave` [2] to `Fortanix EDP` [1].

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. 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.

On the contrary, Fortanix EDP (Enclave Development Platform) applications are just like native Rust applications (4.5). They have a main function, can have multiple threads, and can make network connections.

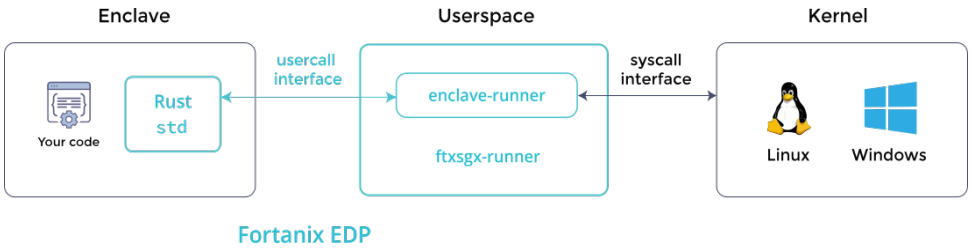

Figure 4.5: EDP architecture
source `https://edp.fortanix.com/docs/concepts/architecture/`

You do not have to write any "untrusted" code that runs outside the enclave since all the features are provided for you. You just need to compile for the `x86_64-fortanix-unknown-sgx` target [89] 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.

Just like a normal OS, the Fortanix EDP defines an API and ABI [90] that applications use to interact with the environment.

The `usercall` API [91] 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.

The native format for SGX enclaves is the SGX stream (SGXS) format [92]. The Rust compiler (normally invoked via Cargo) produces a binary in the ELF format. To run it, it must be converted into the processor's native enclave format.

All enclaves must be signed prior to running. The signing key is a security principal in remote attestation and when using MRSIGNER-based sealing. Also, on platforms without Flexible Launch

Control, the signing key may determine which enclaves can be run in production mode.

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 (for now, support for SGX2 is tracked in [93]). 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).

It is to be noted that Fortanix EDP seems quite robust according to [70], [68], and [94]. Nevertheless, as we can see on the appendix A.5, some default parameters can lead to a situation where an enclave can be in debug mode without specifying it. Obviously, it will be detected by the client but it's something that needs to be checked in a production setup.

INFO 1	Enclaves are unsigned prior to running which is not compliant with Fortanix EDP documentation.	
Category	CWE-345: Insufficient Verification of Data Authenticity	
Rating	**Impact**: Code quality	**Exploitability**: None

Recommendation

Fortanix EDP documentation specifies that all enclaves must be signed prior to running. The signing key is a security principle of remote attestation. Quarkslab auditors recommend using a production signing key as well as storing it securely.

4.4 Code quality

One of the first things we like to do when we encounter a Rust project is to have some ideas about the code quality. To do so several tools from the Rust ecosystem help.

4.4.1 cargo audit

Running `cargo audit` gives some hints on the state of the dependencies.

```
azureuser@blindai-preview:~/blindai-preview$ cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 537 security advisories (from /home/azureuser/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (221 crate dependencies)
Crate:     remove_dir_all
Version:   0.5.3
Title:     Race Condition Enabling Link Following and Time-of-check Time-of-use
↪  (TOCTOU)
Date:      2023-02-24
ID:        RUSTSEC-2023-0018
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0018
Solution:  Upgrade to >=0.8.0
Dependency tree:
remove_dir_all 0.5.3
    tempfile 3.3.0
        multipart 0.18.0
            rouille 3.6.1
                blindai_server 0.0.1

Crate:     buf_redux
Version:   0.8.4
Warning:   unmaintained
Title:     buf_redux is Unmaintained
Date:      2023-01-24
ID:        RUSTSEC-2023-0028
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0028
Dependency tree:
buf_redux 0.8.4
    multipart 0.18.0
        rouille 3.6.1
            blindai_server 0.0.1

Crate:     twoway
Version:   0.1.8
Warning:   unmaintained
Title:     Crate `twoway` deprecated by the author
Date:      2021-05-20
ID:        RUSTSEC-2021-0146
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0146
Dependency tree:
twoway 0.1.8
```

```
    multipart 0.18.0
        rouille 3.6.1
            blindai_server 0.0.1

error: 1 vulnerability found!
warning: 2 allowed warnings found
```

One of the dependencies used by the enclave, `rouille` [3] seems to have some issues. None of them are really problematic. Nevertheless, it would be best to look for other frameworks with better support.

INFO 2	One of the dependencies used by the enclave, `rouille` [3] seems to have some issues.	
Category	CWE-1104: Use of Unmaintained Third Party Components	
Rating	**Impact**: Supply chain	**Exploitability**: None

4.4.2 cargo geiger

`cargo geiger` checks if dependencies used unsafe code. The output is quite long (it can be seen in the appendix A.6). We can notice that several dependencies are using unsafe code. It is to be noted that the `#![forbid(unsafe_code)]` rule forbids unsafe code in the current crate, but it doesn't verify that dependencies do not use unsafe code.

The usage of unsafe code doesn't mean that the code is inherently unsafe, it just highlights the part of the code that can be troublesome.

4.4.3 Improper handling of error conditions

Another thing we notice is that the code used in `BlindAI-preview` seems to be less mature that the code used in `BlindAI`.

For example, the API endpoints are not quite user-friendly. If an error occurs during the request handling, they only return a 500 error code without any diagnostics.

```
$ restish POST https://localhost:9924/delete --rsh-insecure
WARN: Disabling TLS security checks
HTTP/1.1 500 Internal Server Error
Content-Length: 382
Content-Type: application/cbor
Date: Tue, 31 Jan 2023 15:29:23 GMT
Server: tiny-http (Rust)

EOF while parsing a value

Stack backtrace:
   0: <unknown>
   1: <unknown>
   2: <unknown>
   3: <unknown>
   4: <unknown>
   5: <unknown>
   6: <unknown>
   7: <unknown>
   8: <unknown>
   9: <unknown>
  10: <unknown>
  11: <unknown>
  12: <unknown>
  13: <unknown>
  14: <unknown>
  15: <unknown>
  16: <unknown>
  17: <unknown>
  18: <unknown>
  19: <unknown>
  20: <unknown>
```

INFO 3	The code used in `BlindAI` does not properly handles errors or exceptional conditions.	
Category	CWE-703: Improper Check or Handling of Exceptional Conditions	
Rating	**Impact**: Code quality	**Exploitability**: None

4.4.4 cargo clippy

Running `cargo clippy` with some additional lints gives also the same feeling about the code quality as it can be seen in the appendix A.7. For example, some functions are using `unwrap` (which can cause a panic) when they are supposed to handle errors.

INFO 4	Some functions are using `unwrap` (which can cause a panic) when they are supposed to handle errors.	
Category	CWE-676: Use of Potentially Dangerous Function	
Rating	**Impact**: Availability	**Exploitability**: None

4.4.5 Nonce set to 0

When the runner requests a quote from the quoting enclave, it uses a nonce value set to 0.

```
pub fn get_quote(&self, report: Report) -> Result<Vec<u8>> {
    let ecdsa_key_id = self.ecdsa_key_id.clone();
    let quote_result = self
        .aesm_client
        .get_quote_ex(ecdsa_key_id, // key ID
            report.as_ref().to_owned(), // report
            None, // target info
            vec![0; 16]) // nonce
        .unwrap();

    Ok(quote_result.quote().to_vec())
}
```

We did not have time to investigate exactly the impact of this parameter but reusing nonces is not a good practice. In the Intel source code [95], we can find some information about the impact.

> The caller can request a REPORT from the QE using a supplied nonce. This will allow the enclave requesting the quote to verify the QE used to generate the quote. This makes it more difficult for something to spoof a QE and allows the app enclave to catch it earlier. But since the authenticity of the QE lies in the knowledge of the Quote signing key, such spoofing will ultimately be detected by the quote verifier.
> `QE REPORT.ReportData = SHA256(*p_{nonce}||*p_{quote})||0x00)`

INFO 5	When the runner requests a quote from the quoting enclave, it uses a nonce value set to 0 which is not a good practice.	
Category	CWE-323: Reusing a Nonce, Key Pair in Encryption	
Rating	**Impact**: Code quality	**Exploitability**: None

4.4.6 Client

Even if the client is written in Python, most of the code used to verify if the quote and the collateral are correct are done in a library written in cpp with a binding [96]. This library is maintained by Intel [97].

A quick look at this library shows that its main dependencies are `openssl` and `rapidjson`. `rapidjson` is a library from Tencent that is known for not taking security issues [4].

Since Rust is already used for the enclave and since Fortanix EDP has some of the features required to parse the quote and the collateral, from a security perspective it could be best to rewrite parts of the client in Rust to have better control of the dependencies.

INFO 6	Client uses `rapidjson` from Tencent which is known for not taking security issues [4].	
Category	CWE-1357: Reliance on Insufficiently Trustworthy Component	
Rating	**Impact**: Supply chain	**Exploitability**: None

4.4.7 CBOR

Both the untrusted and trusted servers use CBOR to encode data sent and received by the enclave. The library is not maintained anymore [5]. However, the auditors note that Mithril Security internally supports this dependency and uses their own fork of the project.

Going back to using GRPC like it was the case in `BlindAI` may be better.

INFO 7	Usage of CBOR to encode data sent and received by the enclave which is not maintained anymore [5].	
Category	CWE-1104: Use of Unmaintained Third Party Components	
Rating	**Impact**: Supply chain	**Exploitability**: None

5 Threat model and methodology

This section presents the threat model defined by Quarkslab auditors and agreed upon with Mithril Security experts. The threat model definition aims to provide priorities and formal scope for the audit of the BlindAI.

Based on the latter, the auditors defined their security testing methodology, as provided at the end of this section.

5.1 Threat model

The below sections present the threat model defined for the scope of this audit. The threat model includes the following:

- Assets considered sensitive and that must be protected (identified by `AX`);
- Hypothesis related to the BlindAI preview and its environment to take into account for the security tests (identified by `HX`);
- Threat actors that can potentially target BlindAI (identified by `EX`);
- Threats that can be applied by threat actors to retrieve or harm sensitive assets (identified by `TX`).

Each item is detailed in the paragraphs below.

5.1.1 Assets

Assets are elements that must be protected against specific threats in order to ensure the security of the product, for a given perimeter.

A1: Enclave's Identity Private Key

The private key used by the enclave to produce the self-signed certificates in order to establish secure communication channels.

A2: Serialized tensor vector

The serialized vector contains the tensor information. It contains the sensitive data of the client and needs to be processed without any exposure.

A3: AI Model

The model deployed on the VM and on which a client can do inference. It contains the IP data of the client.

A1 and A2 are more sensitive and therefore will require a higher level of protection than A3.

5.1.2 Secure usage hypotheses

This part of the threat model defines some hypotheses made on the usage and environment of the BlindAI.

H1: Secure Client environment

For the client, we assume that its machine is trusted and that the Python client code (and all of its dependencies) has not been altered or tampered.

5.1.3 Security threat actors

This part of the threat model introduces some threat actors that may target some assets of the BlindAI.

E1: Attacker compromising an enclave (MATE attack)

For this threat, we have an attacker that has full control over the OS where the enclave is deployed. This is the ultimate attacker in terms of power (he is able to perform man-at-the-end attacks).

E2: Attacker compromising the network transmission via MITM attacks

For this threat, we have an attacker that is the usual antagonist, performing interception of the transmission between the client and the enclave (man-in-the-middle attacks) and tries to compromise the secure channel established via TLS.

5.1.4 Security Threats

Concerning the aforementioned threat actors, the paragraphs below present some threats that can be enforced by malicious actors to harm or retrieve the BlindAI assets.

Threats related to confidentiality

T1: Asset leakages

- A1: Case where the enclave's private key leaks to the adversary;
- A2: Case where the tensor's vector leaks to the adversary;
- A3: Case where the AI model leaks to the adversary.

T2: Inference results leaking

Case where the adversary is able to intercept the results of the inference done on the enclave.

T3: Enclave's data leaking

Case where the adversary manages to recover information on the enclave's internal data.

Threats related to integrity

T4: Tensor's corruption

Case where the adversary is able to alter the serialized tensor vector's data that is going to be inferred on the enclave.

T5: Model's corruption

Case where the adversary is able to alter the model deployed on the enclave.

T6: Inference results corruption

Case where the adversary is able to alter the results of the inference done on the enclave.

T7: Enclave compromission

Case where the internal data of the enclave is altered by an adversary.

Threats related to authenticity

T8: Malicious enclave trust

Case where the client trusts and attests a malicious enclave.

5.2 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?* To answer this question, and based on the above-mentioned threat model, we observe that an attacker can first attack code running the enclave:

- by fuzzing inputs used by the enclave (for example models);
- by acting as a malicious operation system and conduct iago attacks;
- by acting as a malicious client and try to exhaust the resources used by the enclave.

The auditors note that they 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.

> **Note**
>
> The auditors underline that during the allocated time frame of the audit, iago attacks were not put in place, but some of them were tackled in [70].

Fortanix EDP user-calls interface is very tight and clearly written with an adversary state of mind.

Second, the initial questions can be answered with the fact that an attacker can target the code running on the client:

- we can alter the information returned to the client by the untrusted port (for example the enclave public certificate, the quote or the collateral);
- since the client has all the sensible data, a vulnerability in a library used to parse or validate these information can be catastrophic.

Based on the introduced threat model and methodology, security tests were defined and made on the BlindAI in order to evaluate its resiliency within the defined perimeter.

6 Remote attestation

To gain the trust between a remote provider or producer and the hardware entity, Intel relies on the remote attestation. On the one hand, it ensures to the client that the software is running inside an SGX enclave and on a fully updated system. On the other hand, the attestation guarantees to the hardware of the identity of the software being attested, the execution mode, and if the software was tampered or not. This section presents the remote attestation as observed in BlindAI.

6.1 Introduction

Generally speaking, the goal of Remote Attestation [98] is for a hardware entity or a combination of hardware and software to gain the trust of a remote service provider, such that the service provider can confidently provide the client with the secrets requested. With Intel SGX, Remote Attestation software includes the application's enclave and the Intel-provided Quoting Enclave (QE) and Provisioning Enclave (PvE). The attestation hardware is the Intel SGX-enabled CPU. Remote attestation provides verification for three things: the application's identity, its integrity (that it has not been tampered with), and that it is running securely within an enclave on an Intel SGX-enabled platform. In order to allow a secured SGX executing environment, several Architectural Enclaves (AE) are involved.

One of the AE is the Quoting Enclave (QE). Quoting Enclave is responsible for providing trust in the enclave identity and its execution environment during the remote attestation process. In order to transform a local `REPORT` into a remotely verifiable `QUOTE`, Quoting Enclave uses a platform's unique asymmetric attestation key. The `QUOTE` can then be verified by a remote party using the corresponding public key. Attestation keys are the core assets in the SGX ecosystem. Relying parties trust valid attestation signatures as an Intel-signed certificate that guarantees the platform's authenticity. In order to facilitate SGX provisioning services, Intel operates a dedicated online provisioning infrastructure [99].

Application Enclaves communicate with Architectural Enclaves through Intel SGX AESM, the Application Enclave Service Manager. In more detail, AESM is the system services management agent for SGX-enabled applications. Those services include various components of the SGX system, such as launch approval, remote attestation quote signing, etc. They are implemented as Intel-provided enclaves, i.e. Architectural Enclaves (above-mentioned), and access to those enclaves is given by AESM. AESM runs as a daemon process `aesmd` when the system starts. `aesmd` provides an untrusted API to communicate with Architectural Enclaves using a domain socket, whose path is hard-coded at `/var/run/aesmd/aesm.socket`.

Intel Software Guard Extensions Data Center Attestation Primitives (Intel SGX DCAP [100]) provides SGX attestation support targeted at data centers, cloud services providers and enterprises via elliptic curve digital signature algorithm (ECDSA) to protect keys, isolate encrypted modules, confidential computing, etc. One of the strengths of Intel Software Guard Extensions Data Center Attestation Primitives (Intel SGX DCAP) is that it allows data centers to own their attestation infrastructure. The attestation collateral needed to both generate and verify quotes is contained completely on-premise, which eliminates runtime dependencies on external services and enables

trust decisions to be made in-house. Attestation collateral including TCB Information, Certificate Revocation Lists and QE/QvE Identities. The essence of having such robustness relies on the complete certificate chain, which connects the whole attestation system, from beginning to end. The certification chain allows to verify the identity of the data owner, making sure that no data is tampered with.

6.2 Application in BlindAI

For BlindAI, this means the following. When the enclave starts, it creates two servers: one on port 9923 (named untrusted server) and one on port 9924 (named trusted server) (see Figure 6.1). It also generates a self-signed certificate which is used by the client when it connects to the trusted server and to a runner listening on port 11000.

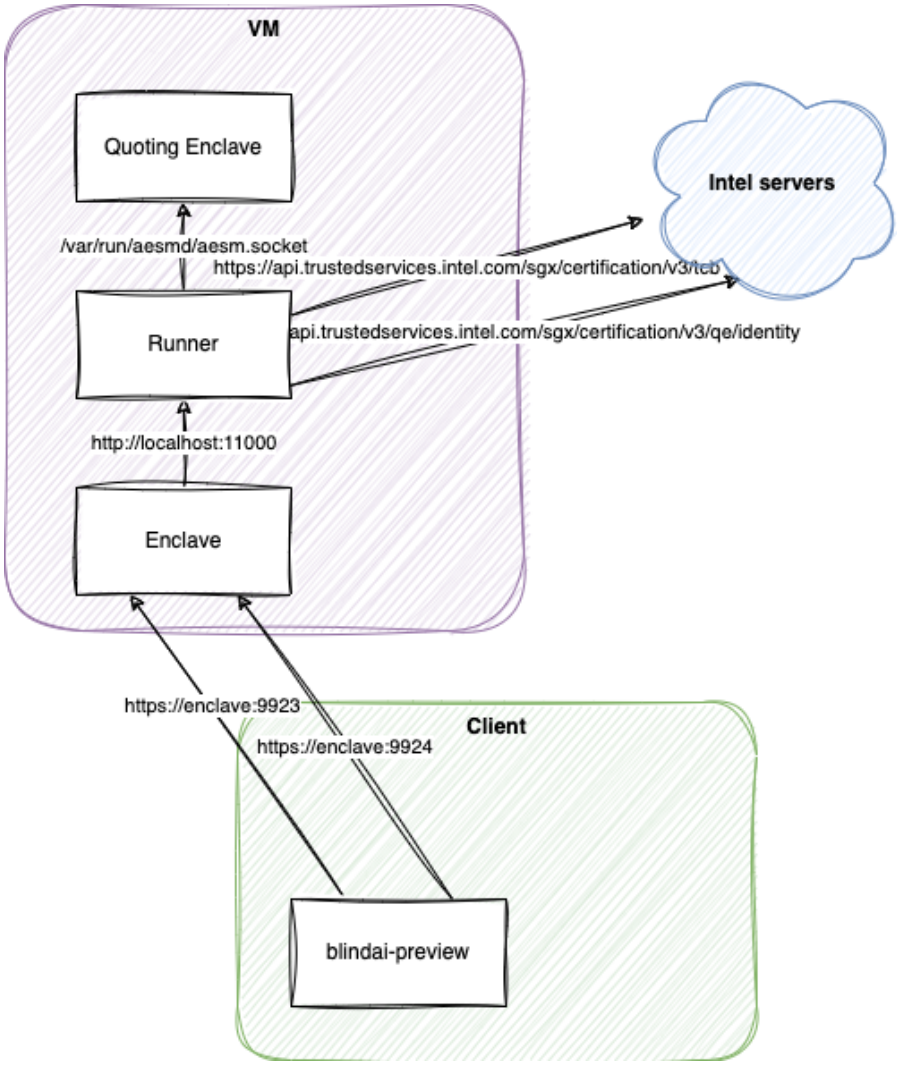

Figure 6.1: Architecture

The untrusted server has 3 endpoints which all return CBOR-encoded data [101]:

- `/`: return the self-signed certificate
- `/quote`: return the quote of the enclave
- `/collateral`: return the collateral

The trusted server also has 3 endpoints:

- `/upload`: to upload a model

- `/run`: to run an inference with a previously stored model
- `/delete`: to delete a model

As of now, all the uploaded models are stored in memory only.

As explained earlier, Fortanix EDP does not support SGX2 at the moment. It means that the enclave memory is allocated when the enclave starts. BlindAI preview does not have a mechanism to offload to the disk when memory consumption is high.

It's quite trivial to have a denial-of-service by sending multiple models.

```
import random

from blindai_preview.client import *
import numpy as np
import onnx
from transformers import DistilBertTokenizer

# For test purpose, we want to avoid setting a TLS reverse proxy on top of
# the untrusted port. We pass the hazmat_http_on_untrusted_port = True argument
# to allow connecting to the untrusted port using plain HTTP instead of HTTPS.
# This option is hazardous therefore it starts with hazmat_
# Those options should generally not be used in production unless you
# have carefully assessed the consequences.
client_v2 = connect(addr="localhost", hazmat_http_on_untrusted_port=True)

model = "distilbert-base-uncased.onnx"
onnx_model = onnx.load(model)

tokenizer = DistilBertTokenizer.from_pretrained("distilbert-base-uncased")
sentence = "I love AI and privacy!"
inputs = {"input": tokenizer(sentence, padding = "max_length", max_length = 8,
↪  return_tensors="np")["input_ids"]}

steps = 13
for i in range(steps):
    iid = random.randint(1, 100000)
    print("======")
    print(f"modifying model {i}: {iid}")
    onnx_model.doc_string = f"{iid}"
    model = onnx_model.SerializeToString()
    print("model length {}".format(len(model)))
    print(f"upload model {i}")
    response = client_v2.upload_model(model=model, model_name="bytes")
    print(f"done")

    print(f"model_id for {i}: {response.model_id}")
    run_response = client_v2.run_model(
        model_id=response.model_id,
        input_tensors=inputs,
    )
```

```
    print("Run successful, got", run_response.output[0].as_numpy())
```

LOW 1	BlindAI does not have a mechanism to offload to the disk when memory consumption is high, which makes a denial-of-service attack possible when multiple models are sent.
Category	CWE-770: Allocation of Resources Without Limits or Throttling
Rating	**Impact**: Denial-of-Service **Exploitability**: High

Recommendation

Despite Denial-of-Service attacks being considered out-of-scope for this assessment by Mithril Security, Quarkslab auditors recommend better control of memory usage.

Obviously, the goal of the client is to verify the enclave with the information returned by the untrusted server before connecting to the trusted server (see Figure 6.2).

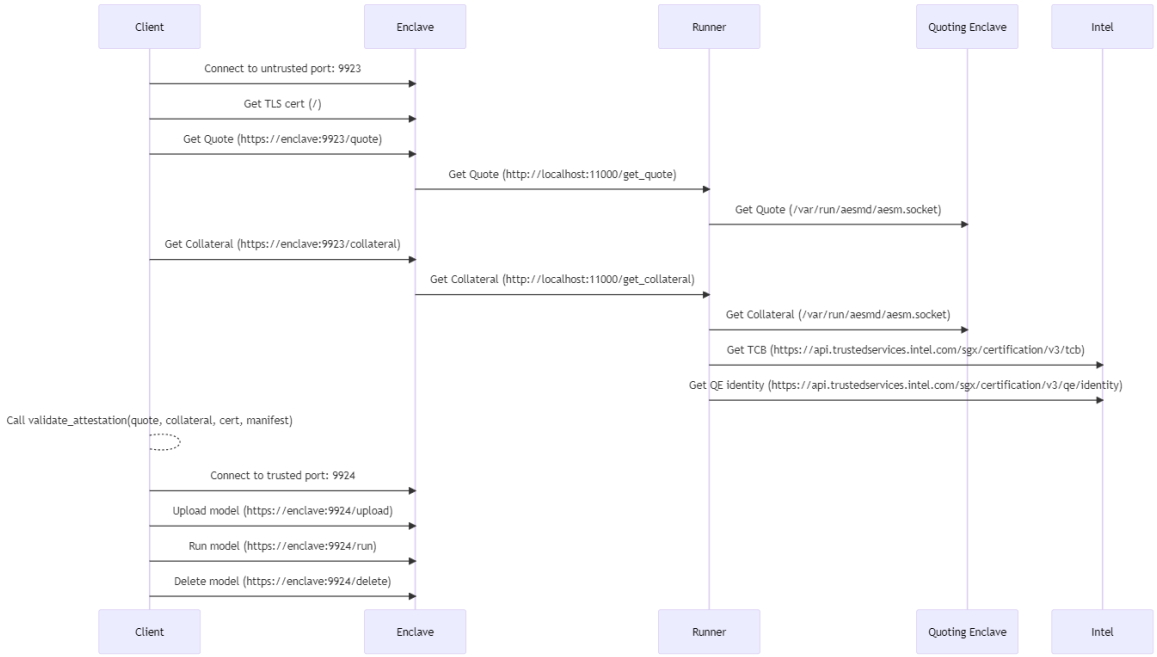

Figure 6.2: Communications

The client first starts by fetching the certificate of the enclave, the quote and the collateral from the untrusted port. BlindAI binds the certificate used by the trusted server to the enclave report by using a scheme similar to the one described in [102]. When an enclave is built and initialized, Intel SGX will generate a cryptographic log of all the build activities, including the following content: code, data, stack, heap, and location of each page within the enclave with the security flags being used. This Enclave Identity, which is a 256-bit hash digest of the log, is stored as `MRENCLAVE` as the enclave's software TCB. Then, all the information needed by the client to check the enclave is stored in a manifest (A.3). As previously explained, quotes (see Figure 7.1) are reports that can be remotely verified.

The verification collateral is the data required by the client to complete the quote verification. It includes:

- The root CA CRL;
- The PCK Cert CRL;
- The PCK Cert CRL signing chain;
- The signing cert chain for the TCBInfo structure;
- The signing cert chain for the QEIdentity structure;
- The TCBInfo structure;
- The QEIdentity structure.

The TCB info (A.2) is a json-encoded structure containing in a way the patch level of the platform TCB. Among other things, it also contains the `FMSPC`. The QE identity (A.1) identifies the enclave used to attest the report of the enclave.

It is to be noted that some information contained in the collateral are also stored in the PCK certificate or the quote.

7 Resiliency tests

This section presents the security tests applied on the BlindAI, with respect to the previously made discovery, cartography, and threat modeling.

7.1 Man-at-the-End

This section presents the tests and related results performed on the BlindAI related to Man-at-the-End attacks, as detailed in Section 5.

7.1.1 RDRAND

RDRAND is an Intel instruction to produce hardware based randomness on Intel on-chip. The entropy source is seeded on-chip as well. The random number generator is NIPS SP 800-90A and FIPS 140-2 compliant. The RDRAND instruction is used by the ring-fortanix crates to produce fresh randomness in `ring-fortanix/src/rand.rs` and the source code contain a version for Intel SGX target.

```
#[cfg(all(target_env = "sgx", target_feature = "rdrand"))]
mod rdrandom {
    use crate::{bssl, error};

    fn rdrand_loop() -> Result<[u8; 8], error::Unspecified> {
        extern "C" {
            fn CRYPTO_rdrand(out: &mut [u8; 8]) -> bssl::Result;
        }

        for _ in 0..10 {
            let mut buf = [0u8; 8];
            match Result::from(unsafe { CRYPTO_rdrand(&mut buf) }) {
                Ok(()) => return Ok(buf),
                Err(_) => continue
            }
        }
        Err(error::Unspecified)
    }

    pub fn fill(dest: &mut [u8]) -> Result<(), error::Unspecified> {
        for dst in dest.chunks_mut(8) {
            let src = rdrand_loop()?;
            let dst_len = dst.len();
            dst.copy_from_slice(&src[..dst_len])
        }

        Ok(())
    }
```

```
}
```

The `CRYPTO_rdrand` function is implemented in assembly in the fips module part of the crate under the path `ring-fortanix/crypto/fipsmodule/randasm/rdrand-86_64`.

```
# CRYPTO_rdrand writes eight bytes of random data from the hardware RNG to
# |out|. It returns one on success or zero on hardware failure.
# int CRYPTO_rdrand(uint8_t out[8]);
.globl    CRYPTO_rdrand
.type    CRYPTO_rdrand,\@function,1
.align    16
CRYPTO_rdrand:
.cfi_startproc
    xorq %rax, %rax
    # This is rdrand %rcx. It sets rcx to a random value and sets the carry
    # flag on success.
    .byte 0x48, 0x0f, 0xc7, 0xf1
    # An add-with-carry of zero effectively sets %rax to the carry flag.
    adcq %rax, %rax
    movq %rcx, 0(%rdi)
    retq
.cfi_endproc
```

In the case of Man-at-the-End attacks, we want to explore the impact of an attacker that has control over the hypervisor and can intercept calls from the BlindAI software to the Intel chip. In such a scenario, the attacker could tamper with RDRAND calls and return biased randomness instead. With such control over randomness, an attacker could have a tremendous impact on the different protocols used in Intel DCAP and remote attestation. In fact, in 2020, a CVE (CVE-2020-0543 [103]) was released where RDRAND was targeted via side-channel attacks, and the researchers were able to extract a full ECDSA key from an SGX enclave after only one signature operation.

After careful code review, we did not find any way to exploit interception at the hypervisor level. Moreover, most cryptographic libraries implement safe-guards for such scenario by combining the output of the random number generator with other inputs, that is then hashed for randomness production inside cryptographic implementations (see for instance OpenSSL nonce generation for ECDSA [104]).

7.1.2 SGX-Step

Most of the relevant attacks on Intel SGX (described in 4.1) are based on a tool called SGX-step: AEPIC attacks, LVI attacks, etc. This framework allows to build a malicious SGX driver, then loading it to the kernel to replace the correct driver.

Following the steps to install SGX-Step, we noticed that the code of the SGX malicious driver had an integer shifting error which did not allow the `insmod` instruction to load the kernel module. This is a somewhat expected issue since the CPU used on the virtual machine is not part of the list of the supported CPUs by SGX-Step. In order to test this framework, a proper fix should be added to the implementation of the malicious driver and could not be done in the time frame of this audit.

7.2 Man-in-the-Middle

This section presents the tests and related results performed on the BlindAI related to Man-in-the-Middle attacks, as detailed in Section 5.

In our threat model, we can, as an attacker able to do a Man-in-the-Middle, modify the following inputs:

- the public enclave certificate;
- the quote;
- the collateral.

Our goal is to make the client trusts us either by bypassing its checks or by exploiting a vulnerability in the parsing.

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.

To verify that the enclave is genuine, after fetching the certificate, the quote, and the collateral, the client does the following steps:

- Set root CA to `Intel_SGX_Provisioning_Certification_RootCA.pem`;
- Call `sgx_dcap_quote_verify` ([105]);
- Check `PCK cert`;
- Check `TCB info`;
- Check `QE identity`;
- Check `Quote`;
- Check enclave report (which contains the SHA256 of enclave cert);
- Check `MRENCLAVE`;
- Check enclave attributes (including debug mode);
- Check enclave `XFRM`;
- Check enclave `MISCSELECT`.

As we can see, the client manifest is paramount to security. If an attacker is able to modify it (for example by allowing a debugging enclave), it is quite simple to bypass all these checks.

Warning

Quarkslab auditors recommend hardening the client in order to provide a way to detect a broken manifest.

It is quite straightforward to replace the enclave certificate with our own self-signed certificate. But since the enclave certificate hash is contained in the quote, any modification will be detected. We did not find any problems regarding the generation of the key pair used by the certificate.

```
$ openssl x509 -inform DER -text -in cert.der
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 5322704629645425893 (0x49de0ba8e657bce5)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = rcgen self signed cert
        Validity
            Not Before: Jan  1 00:00:00 1975 GMT
            Not After : Jan  1 00:00:00 4096 GMT
        Subject: CN = rcgen self signed cert
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:4f:b2:23:f2:dc:d5:96:27:f3:41:3c:b3:fd:00:
                    f9:3d:cb:84:43:6b:3e:91:74:6b:55:be:36:46:f9:
                    3a:83:4d:33:3f:df:84:ee:fb:a3:57:b7:03:52:58:
                    5a:a3:4c:01:72:7f:5b:05:dc:f7:a1:61:7a:1b:d0:
                    f5:3d:4e:c6:a1
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Alternative Name:
                DNS:blindai-srv
    Signature Algorithm: ecdsa-with-SHA256
         30:45:02:21:00:ba:ad:2f:8c:14:86:d7:d8:56:c9:46:5e:f3:
         02:a1:48:17:c5:7a:a8:8b:87:22:bc:80:99:96:32:65:68:5a:
         5a:02:20:4a:41:68:e8:02:99:e2:12:e5:f5:0a:d5:7a:95:c8:
         ca:66:8a:6b:2d:6c:75:7a:28:38:5f:bc:0c:7d:b6:3d:fb
```

If we take a closer look at the quote (See Figure 7.1), the only parameter that is not part of any signature is a field that we called `remaining byte size`.

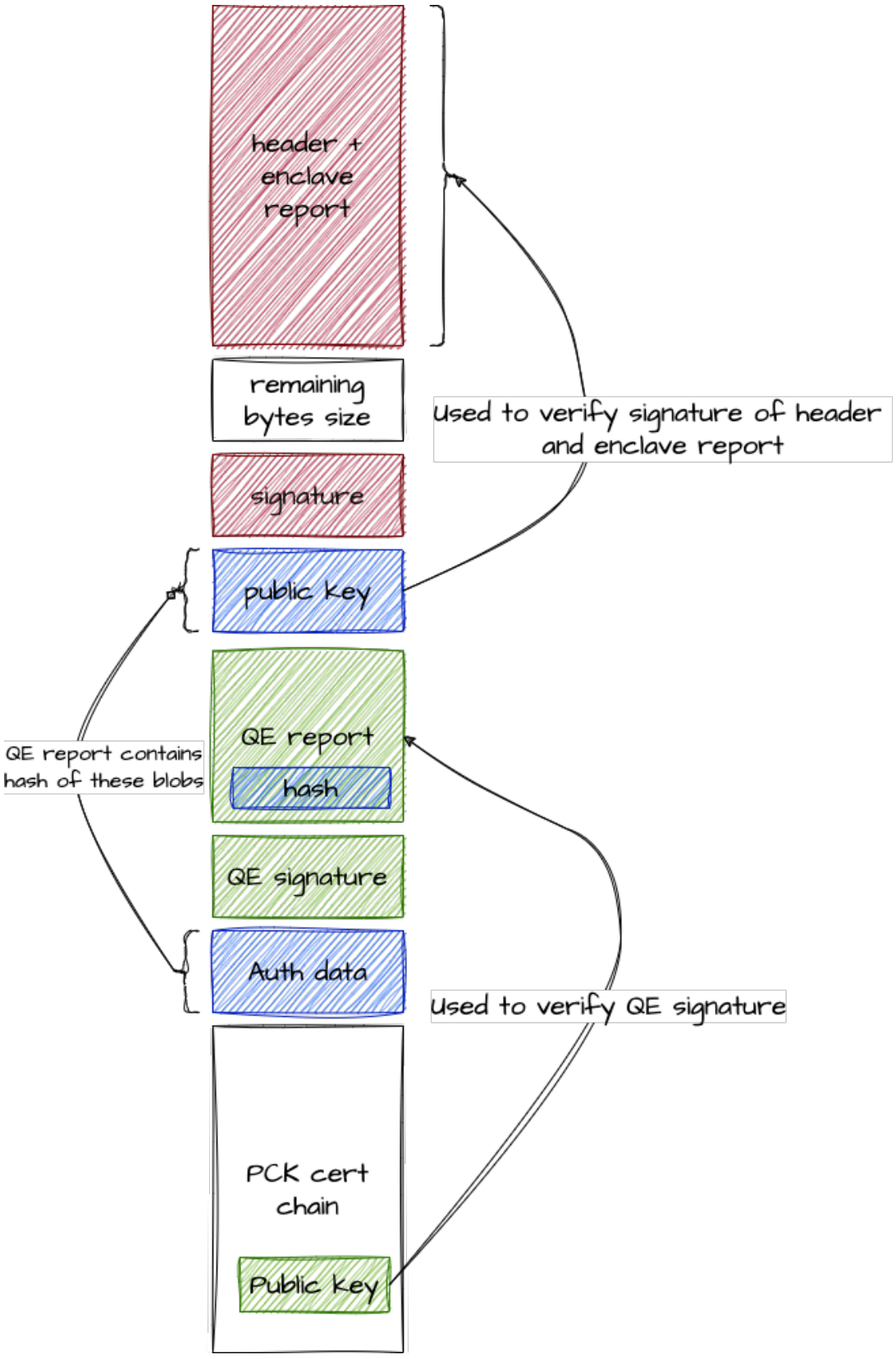

Figure 7.1: Quote

We can not modify this field, the Intel QVL library checks that it has a proper value:

```
$ python proxy.py
Traceback (most recent call last):
File "proxy.py", line 5, in <module>
client_v2 = connect(addr="localhost", hazmat_http_on_untrusted_port=True,
↪  untrusted_port=8923, attested_port=8924)
File "/home/azureuser/blindai-preview/client/blindai_preview/client.py", line 712,
↪  in connect
return BlindAiConnection(*args, **kwargs)
File "/home/azureuser/blindai-preview/client/blindai_preview/client.py", line 527,
↪  in __init__
self._connect_server(
File "/home/azureuser/blindai-preview/client/blindai_preview/client.py", line 572,
↪  in _connect_server
validate_attestation(quote, collateral, cert, manifest_path=manifest_path)
File
↪  "/home/azureuser/blindai-preview/client/blindai_preview/_dcap_attestation.py",
↪  line 150, in validate_attestation
raise QuoteValidationError(
blindai_preview._dcap_attestation.QuoteValidationError: Invalid quote status
↪  STATUS_UNSUPPORTED_QUOTE_FORMAT
```

Modifications to other parts of the quote are also detected during the verification of the quote:

```
$ python proxy.py
Traceback (most recent call last):
File "proxy.py", line 5, in <module>
client_v2 = connect(addr="localhost", hazmat_http_on_untrusted_port=True,
↪  untrusted_port=8923, attested_port=8924)
File "/home/azureuser/blindai-preview/client/blindai_preview/client.py", line 712,
↪  in connect
return BlindAiConnection(*args, **kwargs)
File "/home/azureuser/blindai-preview/client/blindai_preview/client.py", line 527,
↪  in __init__
self._connect_server(
File "/home/azureuser/blindai-preview/client/blindai_preview/client.py", line 572,
↪  in _connect_server
validate_attestation(quote, collateral, cert, manifest_path=manifest_path)
File
↪  "/home/azureuser/blindai-preview/client/blindai_preview/_dcap_attestation.py",
↪  line 150, in validate_attestation
raise QuoteValidationError(
blindai_preview._dcap_attestation.QuoteValidationError: Invalid quote status
↪  STATUS_INVALID_QUOTE_SIGNATURE
```

The verification collateral is the data required by the client to complete the quote verification.

- The root CA CRL;
- The PCK Cert CRL;

- The PCK Cert CRL signing chain;
- The signing cert chain for the TCBInfo structure;
- The signing cert chain for the QEIdentity structure;
- The TCBInfo structure;
- The QEIdentity structure.

Several typical elements from the above:

- Root CA Certificate: Root Certificate, issue Intel SGX-related certificates;
- PCK Platform CA Certificate: Intermediate Certificate, issue Intel SGX PCK certificate and its corresponding revocation list for multi-package platforms;
- PCK Processor CA Certificate: Intermediate Certificate, issue Intel SGX PCK certificate and its corresponding revocation list for single-package platforms;
- PCK Certificate: User Certificate, sign Intel SGX TCB data for Intel platforms (both multi-package and single-package).

Here is the clear relationship among certificates on the certificate chain:

- Root CA $\rightarrow$ PCK platform/Processor CA (Intermediate) $\rightarrow$ PCK Certificate;
- Root certificate issues the intermediate certificate and the intermediate certificate issues the user certificate.

To ensure that the quote is genuine, Intel SDK does the following steps:

- Check if the subject of the `pckCert` is from Intel;
- Check if CRL issuer and `pkcCert` issuer match;
- Check if `pckCert` is not revoked in CRL;
- Check if `TCB info` and quote header have the same `teeType` (SGX or TDX);
- Check if `FMSPC` of `pckCert` and `TCB info` match;
- Check if `PCEID` or `pckCert` and `TCB info` match;
- Check if quote cert is valid;
- Check if quote QE report signature is valid;
- Check if QE report `reportData` is a valid sha256 digest of `attestpubkey` and `qeauthdata`;
- Check if QE identity ID matches a known enclave ID;
- Check if QE identity match QE report;
- Check if quote signature is valid;
- Check if TCB levels in `TDB info` match TCB levels in `pckCert` and in quote.

We did not find a way to bypass all these checks.

8 Conclusion

To conclude, Quarkslab made a state-of-the-art and complete discovery of Intel SGX and related modules linked to the BlindAI. Based on the latter, a cartography and threat modeling were defined in order to focus the security audit on relevant items within the allocated time frame.

Using the defined security perimeter and tests, the audit unveiled some low and mostly informative issues in the codebase, but nothing critical or exploitable in the end.

However, Quarkslab auditors note that the client of the BlindAI preview should also be hardened as, despite being considered out-of-scope for this audit, it can easily become an entrypoint for malicious actors.

Overall, it was a pleasure to work with Mithril Security experts and maintainers on this audit, they were very helpful, security-minded, and willing to make the project more resilient.

Bibliography

[1] *GitHub - fortanix/rust-sgx: The Fortanix Rust Enclave Development Platform*. URL: https://github.com/fortanix/rust-sgx (visited on Mar. 22, 2023) (cit. on pp. 3, 26).

[2] *GitHub - apache/incubator-teaclave-sgx-sdk: Apache Teaclave (incubating) SGX SDK helps developers to write Intel SGX applications in the Rust programming language, and also known as Rust SGX SDK*. URL: https://github.com/apache/incubator-teaclave-sgx-sdk (visited on Mar. 22, 2023) (cit. on pp. 3, 26).

[3] *GitHub - tomaka/rouille: Web framework in Rust*. URL: https://github.com/tomaka/rouille (visited on Mar. 29, 2023) (cit. on pp. 4, 29).

[4] *Where to report security issues? - Issue #1984 - Tencent/rapidjson - GitHub*. URL: https://github.com/Tencent/rapidjson/issues/1984 (visited on Mar. 29, 2023) (cit. on pp. 4, 34).

[5] *GitHub - pyfisch/cbor: CBOR support for serde.* URL: https://github.com/pyfisch/cbor (visited on Mar. 29, 2023) (cit. on pp. 4, 35).

[6] *SGX 101*. URL: https://sgx101.gitbook.io/sgx101/ (visited on Mar. 29, 2023) (cit. on pp. 7, 13).

[7] *Overview of Intel SGX - Part 1 and 2*. URL: https://blog.quarkslab.com/tag/intel-sgx.html (visited on Mar. 29, 2023) (cit. on pp. 7, 13).

[8] *Open Neural Network Exchange · Web page*. URL: https://onnx.ai/ (visited on Mar. 29, 2023) (cit. on p. 10).

[9] *Software Guard Extensions*. URL: https://en.wikipedia.org/wiki/Software_Guard_Extensions (visited on Mar. 29, 2023) (cit. on p. 13).

[10] "*Intel Software Guard Extensions: Innovative Instructions for Next Generation Isolated Execution*". URL: https://www.youtube.com/watch?v=mPT_vJrlHlg (visited on Mar. 29, 2023) (cit. on p. 13).

[11] *Intel SGX Web-Based Training*. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sgx-web-based-training.html (visited on Mar. 29, 2023) (cit. on p. 13).

[12] *Introduction to Intel Software Guard Extensions · Linkedin*. URL: https://www.linkedin.com/pulse/introduction-intel-software-guard-extensions-esmond-dsouza/ (visited on Mar. 29, 2023) (cit. on p. 13).

[13] *Intro to SGX: from HTTP to enclaves · Medium*. URL: https://medium.com/corda/intro-to-sgx-from-http-to-enclaves-1bf38a3bf595 (visited on Mar. 29, 2023) (cit. on p. 13).

[14] Victor Costan and Srinivas Devadas. *Intel SGX Explained*. Cryptology ePrint Archive, Paper 2016/086. 2016 (cit. on p. 13).

[15] *Enclave Layout*. URL: https://gts3.org/pages/enclave-layout.html (visited on Mar. 29, 2023) (cit. on p. 13).

[16] *Intel Software Guard Extensions SGX2*. URL: https://caslab.csl.yale.edu/workshops/hasp2016/HASP16-16_slides.pdf (visited on Mar. 29, 2023) (cit. on p. 13).

[17] *Intel Enhanced Privacy ID (EPID) Security Technology*. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-enhanced-privacy-id-epid-security-technology.html (visited on Mar. 29, 2023) (cit. on p. 14).

[18] *Intel® SGX Data Center Attestation Primitives (Intel® SGX DCAP)*. URL: https://download.01.org/intel-sgx/sgx-dcap/1.9/linux/docs/Intel_SGX_DCAP_ECDSA_Orientation.pdf (visited on Mar. 29, 2023) (cit. on p. 14).
[19] *SGX Basics · INRIA*. URL: https://gitlab.inria.fr/abaud/sgx-basics (visited on Mar. 29, 2023) (cit. on p. 14).
[20] *Gramine · GitHub*. URL: https://github.com/gramineproject/gramine (visited on Mar. 29, 2023) (cit. on p. 14).
[21] Huibo Wang, Pei Wang, Yu Ding, Mingshen Sun, Yiming Jing, Ran Duan, Long Li, Yulong Zhang, Tao Wei, and Zhiqiang Lin. "Towards Memory Safe Enclave Programming with Rust-SGX". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019 (cit. on p. 14).
[22] *Fortanix EDP*. URL: https://edp.fortanix.com/ (visited on Mar. 29, 2023) (cit. on p. 14).
[23] *SGX.Fail*. URL: https://sgx.fail/ (visited on Mar. 29, 2023) (cit. on p. 14).
[24] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. "A Survey of Published Attacks on Intel SGX". In: *ArXiv* (2020) (cit. on p. 14).
[25] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems". In: *2015 IEEE Symposium on Security and Privacy*. 2015 (cit. on p. 16).
[26] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. "Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution". In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017 (cit. on p. 16).
[27] *Stealthy Page Table-Based Attacks on Enclaved Execution Source Code · GitHub*. URL: https://github.com/jovanbulck/sgx-pte (visited on Mar. 29, 2023) (cit. on p. 16).
[28] Jo Van Bulck, Frank Piessens, and Raoul Strackx. "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control". In: *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 2017 (cit. on p. 16).
[29] *SGXStep Framework · jovanbulck/sgx-step · GitHub*. URL: https://github.com/jovanbulck/sgx-step (visited on Mar. 28, 2023) (cit. on pp. 16, 20).
[30] Jo Van Bulck, Frank Piessens, and Raoul Strackx. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic". In: 2018 (cit. on p. 16).
[31] *Nemesis · GitHub*. URL: https://github.com/jovanbulck/nemesis (visited on Mar. 29, 2023) (cit. on p. 16).
[32] Jago Gyselinck, Jo Van Bulck, Frank Piessens, and Raoul Strackx. "Off-Limits: Abusing Legacy x86 Memory Segmentation to Spy on Enclaved Execution". In: *Engineering Secure Software and Systems*. 2018 (cit. on p. 16).
[33] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. "Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017 (cit. on p. 16).
[34] *Leaky Cauldron on the Dark Land Source Code · GitHub*. URL: https://github.com/heartever/SPMattack (visited on Mar. 29, 2023) (cit. on p. 16).
[35] Daniel Moghimi, Jo Van Bulck, Nadia Heninger, Frank Piessens, and Berk Sunar. "CopyCat: Controlled Instruction-Level Attacks on Enclaves". In: *USENIX Security Symposium*. 2020 (cit. on p. 16).

[36] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. “CacheZoom: How SGX Amplifies the Power of Cache Attacks”. In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. 2017 (cit. on p. 16).

[37] *CacheZoom Source Code · GitHub*. URL: https://github.com/vernamlab/CacheZoom (visited on Mar. 29, 2023) (cit. on p. 16).

[38] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. “Cache Attacks on Intel SGX”. In: *Proceedings of the 10th European Workshop on Systems Security*. 2017 (cit. on p. 16).

[39] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. “Malware Guard Extension: Using SGX to Conceal Cache Attacks”. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2017 (cit. on p. 16).

[40] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. “Software Grand Exposure: SGX Cache Attacks Are Practical”. In: *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. 2017 (cit. on p. 16).

[41] Fergus Dall, Gabrielle Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Daniel Moghimi, and Yuval Yarom. “CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks”. In: 2018 (cit. on p. 16).

[42] Daniel Moghimi, Thomas Eisenbarth, and Berk Sunar. “MemJam: A False Dependency Attack Against Constant-Time Crypto Implementations in SGX”. In: *Topics in Cryptology – CT-RSA 2018*. 2018 (cit. on p. 16).

[43] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. “Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution”. In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018 (cit. on p. 16).

[44] *FORESHADOW Source Code · GitHub*. URL: https://github.com/jovanbulck/sgx-step/tree/master/app/foreshadow (visited on Mar. 29, 2023) (cit. on p. 16).

[45] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H. Lai. “SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution”. In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2019 (cit. on p. 17).

[46] *SgxPectre Source Code · GitHub*. URL: https://github.com/OSUSecLab/SgxPectre (visited on Mar. 29, 2023) (cit. on p. 17).

[47] Esmaeil Mohammadian Koruyeh, Khaled N. Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. “Spectre Returns! Speculation Attacks using the Return Stack Buffer”. In: *12th USENIX Workshop on Offensive Technologies (WOOT 18)*. 2018 (cit. on p. 17).

[48] *Spectre Attack Source Code · GitHub*. URL: https://github.com/lsds/spectre-attack-sgx (visited on Mar. 29, 2023) (cit. on p. 17).

[49] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. “Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing”. In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017 (cit. on p. 17).

[50] Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. “BranchScope: A New Side-Channel Attack on Directional Branch Predictor”. In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 2018 (cit. on p. 17).

[51] Tianlin Huo, Xiaoni Meng, Wenhao Wang, Chunliang Hao, Pei Zhao, Jian Zhai, and Mingshu Li. “Bluethunder: A 2-level Directional Predictor Based Side-Channel Attack against SGX”. In: 2019 (cit. on p. 17).

[52] Stephan van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. “RIDL: Rogue In-Flight Data Load”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019 (cit. on p. 17).

[53] *RIDL Source Code · GitHub*. URL: https://github.com/vusec/ridl (visited on Mar. 29, 2023) (cit. on p. 17).

[54] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. “ZombieLoad: Cross-Privilege-Boundary Data Sampling”. In: *CCS*. 2019 (cit. on p. 17).

[55] *ZombieLoad Web Page*. URL: https://zombieloadattack.com/ (visited on Mar. 29, 2023) (cit. on p. 17).

[56] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. “CacheOut: Leaking Data on Intel CPUs via Cache Evictions”. In: *S&P*. 2021 (cit. on pp. 17, 19).

[57] *CacheOut and SGXAxe attacks*. URL: https://sgaxe.com/ (visited on Mar. 29, 2023) (cit. on pp. 17, 19).

[58] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. *SGAxe: How SGX Fails in Practice*. https://sgaxeattack.com/. 2020 (cit. on p. 17).

[59] Hany Ragab, Alyssa Milburn, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. “CrossTalk: Speculative Data Leaks Across Cores Are Real”. In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021 (cit. on p. 17).

[60] *CROSSTALK Source Code · GitHub*. URL: https://github.com/tristan-hornetz/crosstalk (visited on Mar. 29, 2023) (cit. on p. 17).

[61] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. “Plundervolt: Software-based Fault Injection Attacks against Intel SGX”. In: *2020 IEEE Symposium on Security and Privacy (SP)*. 2020 (cit. on p. 17).

[62] *Plundervolt Source Code · GitHub*. URL: https://github.com/KitMurdock/plundervolt (visited on Mar. 29, 2023) (cit. on p. 17).

[63] Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. “V0LTpwn: Attacking x86 Processor Integrity from Software”. In: *29th USENIX Security Symposium (USENIX Security 20)*. 2020 (cit. on p. 17).

[64] *V0LTpwn Source Code · GitHub*. URL: https://github.com/zkenjar/v0ltpwn (visited on Mar. 29, 2023) (cit. on p. 17).

[65] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. “AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves”. In: *Computer Security – ESORICS 2016*. 2016 (cit. on p. 18).

[66] Jose Rodrigo Sanchez Vicarte, Benjamin Schreiber, Riccardo Paccagnella, and Christopher W. Fletcher. “Game of Threads: Enabling Asynchronous Poisoning Attacks”. In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020 (cit. on p. 18).

[67] *Game of Threads Source Code · GitHub*. URL: https://github.com/FPSG-UIUC/hogwild_pytorch (visited on Mar. 29, 2023) (cit. on p. 18).

[68] Jinhua Cui, Jason Zhijingcheng Yu, Shweta Shinde, Prateek Saxena, and Zhiping Cai. “SmashEx: Smashing SGX Enclaves Using Exceptions”. In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2021 (cit. on pp. 18, 27).

[69] *SmashEx PoC Exploits for Intel SGX Frameworks based on Intel SGX SDK · GitHub*. URL: https://github.com/cimcs/poc-exploits-of-smashex (visited on Mar. 29, 2023) (cit. on p. 18).

[70] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D. Garcia, and Frank Piessens. "A Tale of Two Worlds: Assessing the Vulnerability of Enclave Shielding Runtimes". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019 (cit. on pp. 18, 27, 39).

[71] *A tale of two worlds Source Code · GitHub*. URL: https://github.com/jovanbulck/0xbadc0de (visited on Mar. 29, 2023) (cit. on p. 18).

[72] Andrea Biondo, Mauro Conti, Lucas Davi, Tommaso Frassetto, and Ahmad-Reza Sadeghi. "The Guard's Dilemma: Efficient Code-Reuse Attacks Against Intel SGX". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018 (cit. on p. 18).

[73] Dimitrios Skarlatos, Mengjia Yan, Bhargava Gopireddy, Read Sprabery, Josep Torrellas, and Christopher W. Fletcher. "MicroScope: Enabling Microarchitectural Replay Attacks". In: *Proceedings of the 46th International Symposium on Computer Architecture*. 2019 (cit. on p. 18).

[74] *MicroScope Source code · GitHub*. URL: https://github.com/dskarlatos/MicroScope (visited on Mar. 29, 2023) (cit. on p. 18).

[75] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. "PLATYPUS: Software-based Power Side-Channel Attacks on x86". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021 (cit. on p. 18).

[76] *Platypus: with great power comes great leakage · Web page*. URL: https://platypusattack.com/ (visited on Mar. 29, 2023) (cit. on p. 18).

[77] *RDRAND · Wikipedia*. URL: https://en.wikipedia.org/wiki/RDRAND (visited on Mar. 29, 2023) (cit. on p. 18).

[78] *Load Injection Value*. URL: https://lviattack.eu/ (visited on Mar. 29, 2023) (cit. on p. 19).

[79] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and Frank Piessens. "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection". In: *41th IEEE Symposium on Security and Privacy (S&P'20)*. 2020 (cit. on p. 19).

[80] *AEPIC Leak*. URL: https://aepicleak.com/ (visited on Mar. 29, 2023) (cit. on p. 19).

[81] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. "ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture". In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022 (cit. on p. 19).

[82] Jo Van Bulck, Frank Piessens, and Raoul Strackx. "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control". In: *2nd Workshop on System Software for Trusted Execution (SysTEX)*. 2017 (cit. on p. 19).

[83] *SGXStep Framework · jovanbulck/sgx-step · GitHub*. URL: https://github.com/jovanbulck/sgx-step (visited on Mar. 28, 2023) (cit. on p. 20).

[84] Tobias Cloosters, Johannes Willbold, Thorsten Holz, and Lucas Davi. "SGXFuzz: Efficiently Synthesizing Nested Structures for SGX Enclave Fuzzing". In: *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*. 2022 (cit. on p. 20).

[85] *SGXFuzz Source Code · uni-due-syssec/sgxfuzz · GitHub*. URL: https://github.com/uni-due-syssec/sgxfuzz (visited on Mar. 28, 2023) (cit. on p. 20).

[86] *justfile overview · mithril-security/blindai-preview/ · GitHub*. URL: https://github.com/mithril-security/blindai-preview/tree/v0.0.2#justfile-overview (visited on Jan. 25, 2023) (cit. on p. 21).

[87] *Installing overview · blindai/ · Pypi*. URL: https://pypi.org/project/blindai/ (visited on Mar. 28, 2023) (cit. on p. 21).
[88] *install client sdk · mithril-security/blindai-preview/ · GitHub*. URL: https://github.com/mithril-security/blindai-preview/tree/v0.0.2#getting-started (visited on Jan. 25, 2023) (cit. on p. 21).
[89] *rust/library/std/src/sys/sgx at master · rust-lang/rust · GitHub*. URL: https://github.com/rust-lang/rust/tree/master/library/std/src/sys/sgx (visited on Mar. 22, 2023) (cit. on p. 26).
[90] *rust-sgx/FORTANIX-SGX-ABI.md at master · fortanix/rust-sgx · GitHub*. URL: https://github.com/fortanix/rust-sgx/blob/master/doc/FORTANIX-SGX-ABI.md (visited on Mar. 22, 2023) (cit. on p. 26).
[91] *rust/mod.rs at master · rust-lang/rust · GitHub*. URL: https://github.com/rust-lang/rust/blob/master/library/std/src/sys/sgx/abi/usercalls/mod.rs (visited on Mar. 22, 2023) (cit. on p. 26).
[92] *rust-sgx/SGXS.md at master · fortanix/rust-sgx · GitHub*. URL: https://github.com/fortanix/rust-sgx/blob/master/doc/SGXS.md (visited on Mar. 22, 2023) (cit. on p. 26).
[93] *SGX2 support - Issue #104 - fortanix/rust-sgx - GitHub*. URL: https://github.com/fortanix/rust-sgx/issues/104 (visited on Mar. 22, 2023) (cit. on p. 27).
[94] *Verifying Correctness of Intel SGX Software Mitigations in Fortanix EDP | Fortanix*. URL: https://www.fortanix.com/blog/2022/07/verifying-correctness-of-intel-sgx-software-mitigations-in-fortanix-edp (visited on Mar. 22, 2023) (cit. on p. 27).
[95] *linux-sgx/sgx_uae_quote_ex.h at 26c458905b72e66db7ac1feae04b43461ce1b76f - intel/linux-sgx - GitHub*. URL: https://github.com/intel/linux-sgx/blob/26c458905b72e66db7ac1feae04b43461ce1b76f/common/inc/sgx_uae_quote_ex.h#L158 (visited on Mar. 29, 2023) (cit. on p. 33).
[96] *sgx-dcap-quote-verify-python - PyPI*. URL: https://pypi.org/project/sgx-dcap-quote-verify-python/ (visited on Mar. 29, 2023) (cit. on p. 34).
[97] *SGXDataCenterAttestationPrimitives/QuoteVerification/QVL at master - intel/SGXDataCenterAttestationPrimitives - GitHub*. URL: https://github.com/intel/SGXDataCenterAttestationPrimitives/tree/master/QuoteVerification/QVL (visited on Mar. 29, 2023) (cit. on p. 34).
[98] *Attestation - SGX 101*. URL: https://sgx101.gitbook.io/sgx101/sgx-bootstrap/attestation (visited on Mar. 15, 2023) (cit. on p. 40).
[99] *Intel® Trusted Services API Management Developer Portal*. URL: https://api.portal.trustedservices.intel.com/documentation#register-platform (visited on Mar. 15, 2023) (cit. on p. 40).
[100] *Demystify Remote Attestation: Explore the DCAP Certificate Chain - Safeheron*. URL: https://blog.safeheron.com/blog/insights/safeheron-originals/demystify-remote-attestation-explore-the-dcap-certificate-chain (visited on Mar. 15, 2023) (cit. on p. 40).
[101] *CBOR — Concise Binary Object Representation | Overview*. URL: https://cbor.io/ (visited on Mar. 15, 2023) (cit. on p. 42).
[102] *Integrating Intel SGX Remote Attestation with Transport Layer Security*. URL: https://arxiv.org/ftp/arxiv/papers/1801/1801.05863.pdf (visited on Mar. 15, 2023) (cit. on p. 45).

[103] *CVE - Incomplete cleanup from specific special register read operations*. URL: https://nvd.nist.gov/vuln/detail/CVE-2020-0543 (visited on Mar. 28, 2023) (cit. on p. 47).

[104] *openssl/openssl/blob/master/crypto/bn/bn_rand.c at ligne 249 · openssl/openssl/blob/master/crypto/bn/ · OpenSSL*. URL: https://github.com/openssl/openssl/blob/master/crypto/bn/bn_rand.c (visited on Mar. 28, 2023) (cit. on p. 47).

[105] *sgx-dcap-quote-verify-python/_core_wrapper.py at v0.0.2 · mithril-security/sgx-dcap-quote-verify-python · GitHub*. URL: https://github.com/mithril-security/sgx-dcap-quote-verify-python/blob/v0.0.2/sgx_dcap_quote_verify/_core_wrapper.py#L65 (visited on Mar. 15, 2023) (cit. on p. 49).

[106] *Debugging Manual | Rust EDP*. URL: https://edp.fortanix.com/docs/tasks/debugging/ (visited on Mar. 28, 2023) (cit. on p. 66).

[107] *GDB script doesn't work with Linux Augusta loader · Issue #301 · fortanix/rust-sgx · GitHub*. URL: https://github.com/fortanix/rust-sgx/issues/301 (visited on Mar. 28, 2023) (cit. on p. 66).

Appendix A

Appendix

A.1 Example of QE identity

The QE identity (A.1) identifies the enclave used to attest the report of the enclave. This appendix provides an example of the QE identity as observed during the assessment.


```
{
  "enclaveIdentity": {
    "attributes": "11000000000000000000000000000000",
    "attributesMask": "FBFFFFFFFFFFFFFF0000000000000000",
    "id": "QE",
    "issueDate": "2021-07-01T23:45:00Z",
    "isvprodid": 1,
    "miscselect": "00000000",
    "miscselectMask": "FFFFFFFF",
    "mrsigner":
↪   "8C4F5775D796503E96137F77C68A829A0056AC8DED70140B081B094490C57BFF",
    "nextUpdate": "2021-07-31T23:45:00Z",
    "tcbEvaluationDataNumber": 10,
    "tcbLevels": [
      {
        "tcb": {
          "isvsvn": 5
        },
        "tcbDate": "2020-11-11T00:00:00Z",
        "tcbStatus": "UpToDate"
      },
      {
        "tcb": {
          "isvsvn": 4
        },
        "tcbDate": "2019-11-13T00:00:00Z",
        "tcbStatus": "OutOfDate"
      },
      {
        "tcb": {
          "isvsvn": 2
        },
        "tcbDate": "2019-05-15T00:00:00Z",
        "tcbStatus": "OutOfDate"
      },
      {
        "tcb": {
          "isvsvn": 1
```

```
        },
        "tcbDate": "2018-08-15T00:00:00Z",
        "tcbStatus": "OutOfDate"
      }
    ],
    "version": 2
  },
  "signature": "c6142237e9f3694d8bad441b9dce64c6a5b6eb8e160647ba8ed0543adca7732c0
↪  db088b8b8307e9350ee1511a366a79e32a3db3d1a1c90cbbdccb386f0104211"
}
```


A.2 Example of TCB info

The TCB info (A.2) is a json-encoded structure containing in a way the patch level of the platform TCB. The appendix below provides an example observed during the assessment.


```
{
  "signature": "b956dfd032434a1d638b11d007a947c5bf0573a320252b6589b9d985421cfb0a6
↪  d748a2b9d05c3e29b65c38e41ee2e17522ec92bfa13eb4009a7cf99b3831f4d",
  "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"
      },
      {
```

```
        "tcb": {
          "pcesvn": 10,
          "sgxtcbcomp01svn": 3,
          "sgxtcbcomp02svn": 3,
          "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": "2020-06-10T00:00:00Z",
        "tcbStatus": "OutOfDate"
      },
      {
        "tcb": {
          "pcesvn": 5,
          "sgxtcbcomp01svn": 3,
          "sgxtcbcomp02svn": 3,
          "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": "2018-01-04T00:00:00Z",
        "tcbStatus": "OutOfDate"
      }
    ],
    "tcbType": 0,
    "version": 2
  }
}
```

A.3 Example of client manifest

The appendix below provides an example of the client manifest as observed during the assessment.

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

# Enclave measurement
# MRENCLAVE represents the enclave's contents and build process
mr_enclave = "2bf473aa6a74217707635b95704ccda29ee9b2a2c2906b1abbee5c8338c4bc3b"

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

# 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
# See <https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia
↪  -32-architectures-software-developer-vol-3d-part-4-manual.pdf>

# 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
# Intel documentation
# > Legal values for SECS.ATTRIBUTES.XFRM conform to these requirements:
# >  * XFRM[1:0] must be set to 0x3.
attributes_xfrm_hex = "0x3"
# ATTRIBUTEMASK.XFRM default : ~0x0 = 0xffffffffffffffff
# For our usage we need to allow CPU features related to AVX2/AVX512
# So we disable the checks for the following bitfield :
#  * 2 => AVX (AVX enable, and XSAVE feature set can be used to manage YMM regs)
#  * 5 => opmask (AVX-512 enable, and XSAVE feature set can be used for AVX
↪  opmask, AKA k-mask, regs)
#  * 6 => ZMM_hi256 (AVX-512 enable, and XSAVE feature set can be used for
↪  upper-halves of the lower ZMM regs)
#  * 7 => Hi16_ZMM (AVX-512 enable, and XSAVE feature set can be used for the
↪  upper ZMM regs)
# Compute mask : ~(1 << 2 | 1 << 5 | 1 << 6 | 1 << 7) == 0xffffffffffffff1b
# May need to be change according to the platform it's running on
# but be careful it can impact the security of the enclave
# You can always disable the uneeded feature at launch to
```

```
# satisfy the mask, so there is actually very few reasons to change this mask :
attributes_mask_xfrm_hex = "0xfffffffffffffff1b"

# From <https://01.org/sites/default/files/documentation/intel_sgx_sdk_developer_
↪  reference_for_linux_os_pdf.pdf>
# > MiscSelect and MiscMask are for future functional extension.
# > Currently, MiscSelect must be 0.
# > Otherwise the corresponding enclave may not be loaded successfully
# Note this is changing with SGX2, the MISCSELECT[0] bit will indicate
# whether exception information on #GP or #PF that occurred inside
# an enclave can be written to the EXINFO structure
misc_select_hex = "0x0"
misc_mask_hex = "0xffffffff"
```

A.4 Accessing enclave memory from a debugger

All the following tests have been done with a dummy enclave using Fortanix quickstart.

To debug, the auditors followed these instructions in [106].

```
$ gdb --args ftxsgx-runner <path_to_sgxs_file>
(gdb) source <path_to_gdb.py>
```

It is to be noted that the script provided by Fortanix is not working with Montgomery SGX driver (as highlighted in [107]).

When we look at the mappings, we noticed that some areas belong to the enclave:

```
(gdb) info proc mappings
  process 354007
  Mapped address spaces:

  Start Addr           End Addr       Size     Offset objfile
  0x555555554000     0x555555570000    0x1c000        0x0
↪  /home/azureuser/.cargo/bin/ftxsgx-runner
  0x555555570000     0x5555556c4000   0x154000    0x1c000
↪  /home/azureuser/.cargo/bin/ftxsgx-runner
  0x5555556c4000     0x55555572e000    0x6a000   0x170000
↪  /home/azureuser/.cargo/bin/ftxsgx-runner
  0x55555572e000     0x55555573e000    0x10000   0x1d9000
↪  /home/azureuser/.cargo/bin/ftxsgx-runner
  0x55555573e000     0x55555573f000     0x1000   0x1e9000
↪  /home/azureuser/.cargo/bin/ftxsgx-runner
  0x55555573f000     0x555555781000    0x42000        0x0 [heap]
  0x7fffd8000000     0x7fffd8021000    0x21000        0x0
  0x7fffd8021000     0x7fffdc000000  0x3fdf000        0x0
  0x7fffe0000000     0x7fffe0021000    0x21000        0x0
  0x7fffe0021000     0x7fffe4000000  0x3fdf000        0x0
```

```
  0x7fffe4000000     0x7fffe4021000    0x21000        0x0
  0x7fffe4021000     0x7fffe8000000  0x3fdf000        0x0
  0x7fffe8000000     0x7fffe8021000    0x21000        0x0
  0x7fffe8021000     0x7fffec000000  0x3fdf000        0x0
  0x7fffec000000     0x7fffec021000    0x21000        0x0
  0x7fffec021000     0x7ffff0000000  0x3fdf000        0x0
  0x7ffff0000000     0x7ffff000f000     0xf000        0x0 /dev/sgx_enclave
  0x7ffff000f000     0x7ffff0034000    0x25000        0x0 /dev/sgx_enclave
  0x7ffff0034000     0x7ffff2038000  0x2004000        0x0 /dev/sgx_enclave
  0x7ffff2038000     0x7ffff2048000    0x10000  0x2771000 /dev/zero (deleted)
  0x7ffff2048000     0x7ffff206b000    0x23000        0x0 /dev/sgx_enclave
  0x7ffff206b000     0x7ffff207b000    0x10000  0x27a4000 /dev/zero (deleted)
  0x7ffff207b000     0x7ffff209e000    0x23000        0x0 /dev/sgx_enclave
  0x7ffff209e000     0x7ffff20ae000    0x10000  0x27d7000 /dev/zero (deleted)
  0x7ffff20ae000     0x7ffff20d1000    0x23000        0x0 /dev/sgx_enclave
  0x7ffff20d1000     0x7ffff20e1000    0x10000  0x280a000 /dev/zero (deleted)
  0x7ffff20e1000     0x7ffff2104000    0x23000        0x0 /dev/sgx_enclave
  0x7ffff2104000     0x7ffff4000000  0x1efc000  0x283d000 /dev/zero (deleted)
  0x7ffff6eb9000     0x7ffff6eba000     0x1000        0x0
  0x7ffff6eba000     0x7ffff6ebc000     0x2000        0x0
  0x7ffff6ebc000     0x7ffff6ebd000     0x1000        0x0
  0x7ffff6ebd000     0x7ffff70bd000   0x200000        0x0
  0x7ffff70bd000     0x7ffff70be000     0x1000        0x0
  0x7ffff70be000     0x7ffff70c0000     0x2000        0x0
  0x7ffff70c0000     0x7ffff70c1000     0x1000        0x0
  0x7ffff70c1000     0x7ffff72c1000   0x200000        0x0
  0x7ffff72c1000     0x7ffff72c2000     0x1000        0x0
  0x7ffff72c2000     0x7ffff72c4000     0x2000        0x0
  0x7ffff72c4000     0x7ffff72c5000     0x1000        0x0
  0x7ffff72c5000     0x7ffff74c5000   0x200000        0x0
  0x7ffff74c5000     0x7ffff74c6000     0x1000        0x0
  0x7ffff74c6000     0x7ffff76c6000   0x200000        0x0
  0x7ffff76c6000     0x7ffff76c7000     0x1000        0x0
  0x7ffff76c7000     0x7ffff78c9000   0x202000        0x0
  0x7ffff78c9000     0x7ffff78eb000    0x22000        0x0
↪  /usr/lib/x86_64-linux-gnu/libc-2.31.so
  0x7ffff78eb000     0x7ffff7a63000   0x178000    0x22000
↪  /usr/lib/x86_64-linux-gnu/libc-2.31.so
  0x7ffff7a63000     0x7ffff7ab1000    0x4e000   0x19a000
↪  /usr/lib/x86_64-linux-gnu/libc-2.31.so
  0x7ffff7ab1000     0x7ffff7ab5000     0x4000   0x1e7000
↪  /usr/lib/x86_64-linux-gnu/libc-2.31.so
  0x7ffff7ab5000     0x7ffff7ab7000     0x2000   0x1eb000
↪  /usr/lib/x86_64-linux-gnu/libc-2.31.so
  0x7ffff7ab7000     0x7ffff7abb000     0x4000        0x0
  0x7ffff7abb000     0x7ffff7ac8000     0xd000        0x0
↪  /usr/lib/x86_64-linux-gnu/libm-2.31.so
  0x7ffff7ac8000     0x7ffff7b6f000    0xa7000     0xd000
↪  /usr/lib/x86_64-linux-gnu/libm-2.31.so
  0x7ffff7b6f000     0x7ffff7c08000    0x99000    0xb4000
↪  /usr/lib/x86_64-linux-gnu/libm-2.31.so
```

```
  0x7ffff7c08000     0x7ffff7c09000     0x1000   0x14c000
↪  /usr/lib/x86_64-linux-gnu/libm-2.31.so
  0x7ffff7c09000     0x7ffff7c0a000     0x1000   0x14d000
↪  /usr/lib/x86_64-linux-gnu/libm-2.31.so
  0x7ffff7c0a000     0x7ffff7c0c000     0x2000        0x0
  0x7ffff7c0c000     0x7ffff7c12000     0x6000        0x0
↪  /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
  0x7ffff7c12000     0x7ffff7c23000    0x11000     0x6000
↪  /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
  0x7ffff7c23000     0x7ffff7c29000     0x6000    0x17000
↪  /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
  0x7ffff7c29000     0x7ffff7c2a000     0x1000    0x1c000
↪  /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
  0x7ffff7c2a000     0x7ffff7c2b000     0x1000    0x1d000
↪  /usr/lib/x86_64-linux-gnu/libpthread-2.31.so
  0x7ffff7c2b000     0x7ffff7c2f000     0x4000        0x0
  0x7ffff7c2f000     0x7ffff7c31000     0x2000        0x0
↪  /usr/lib/x86_64-linux-gnu/librt-2.31.so
  0x7ffff7c31000     0x7ffff7c35000     0x4000     0x2000
↪  /usr/lib/x86_64-linux-gnu/librt-2.31.so
  0x7ffff7c35000     0x7ffff7c37000     0x2000     0x6000
↪  /usr/lib/x86_64-linux-gnu/librt-2.31.so
  0x7ffff7c37000     0x7ffff7c38000     0x1000     0x7000
↪  /usr/lib/x86_64-linux-gnu/librt-2.31.so
  0x7ffff7c38000     0x7ffff7c39000     0x1000     0x8000
↪  /usr/lib/x86_64-linux-gnu/librt-2.31.so
  0x7ffff7c39000     0x7ffff7c3c000     0x3000        0x0
↪  /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
  0x7ffff7c3c000     0x7ffff7c4e000    0x12000     0x3000
↪  /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
  0x7ffff7c4e000     0x7ffff7c52000     0x4000    0x15000
↪  /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
  0x7ffff7c52000     0x7ffff7c53000     0x1000    0x18000
↪  /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
  0x7ffff7c53000     0x7ffff7c54000     0x1000    0x19000
↪  /usr/lib/x86_64-linux-gnu/libgcc_s.so.1
  0x7ffff7c54000     0x7ffff7ccc000    0x78000        0x0
↪  /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
  0x7ffff7ccc000     0x7ffff7e68000   0x19c000    0x78000
↪  /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
  0x7ffff7e68000     0x7ffff7ef9000    0x91000   0x214000
↪  /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
  0x7ffff7ef9000     0x7ffff7f25000    0x2c000   0x2a4000
↪  /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
  0x7ffff7f25000     0x7ffff7f27000     0x2000   0x2d0000
↪  /usr/lib/x86_64-linux-gnu/libcrypto.so.1.1
  0x7ffff7f27000     0x7ffff7f2b000     0x4000        0x0
  0x7ffff7f2b000     0x7ffff7f47000    0x1c000        0x0
↪  /usr/lib/x86_64-linux-gnu/libssl.so.1.1
  0x7ffff7f47000     0x7ffff7f96000    0x4f000    0x1c000
↪  /usr/lib/x86_64-linux-gnu/libssl.so.1.1
```

```
  0x7ffff7f96000     0x7ffff7fb0000    0x1a000    0x6b000
↪  /usr/lib/x86_64-linux-gnu/libssl.so.1.1
  0x7ffff7fb0000     0x7ffff7fb1000     0x1000    0x85000
↪  /usr/lib/x86_64-linux-gnu/libssl.so.1.1
  0x7ffff7fb1000     0x7ffff7fba000     0x9000    0x85000
↪  /usr/lib/x86_64-linux-gnu/libssl.so.1.1
  0x7ffff7fba000     0x7ffff7fbe000     0x4000    0x8e000
↪  /usr/lib/x86_64-linux-gnu/libssl.so.1.1
  0x7ffff7fbe000     0x7ffff7fbf000     0x1000        0x0
↪  /usr/lib/x86_64-linux-gnu/libdl-2.31.so
  0x7ffff7fbf000     0x7ffff7fc1000     0x2000     0x1000
↪  /usr/lib/x86_64-linux-gnu/libdl-2.31.so
  0x7ffff7fc1000     0x7ffff7fc2000     0x1000     0x3000
↪  /usr/lib/x86_64-linux-gnu/libdl-2.31.so
  0x7ffff7fc2000     0x7ffff7fc3000     0x1000     0x3000
↪  /usr/lib/x86_64-linux-gnu/libdl-2.31.so
  0x7ffff7fc3000     0x7ffff7fc4000     0x1000     0x4000
↪  /usr/lib/x86_64-linux-gnu/libdl-2.31.so
  0x7ffff7fc4000     0x7ffff7fc6000     0x2000        0x0
  0x7ffff7fc6000     0x7ffff7fc7000     0x1000        0x0
  0x7ffff7fc7000     0x7ffff7fc9000     0x2000        0x0
  0x7ffff7fc9000     0x7ffff7fca000     0x1000        0x0
  0x7ffff7fca000     0x7ffff7fcc000     0x2000        0x0
  0x7ffff7fcc000     0x7ffff7fcd000     0x1000        0x0
  0x7ffff7fcd000     0x7ffff7fcf000     0x2000        0x0
  0x7ffff7fcf000     0x7ffff7fd0000     0x1000        0x0
↪  /usr/lib/x86_64-linux-gnu/ld-2.31.so
  0x7ffff7fd0000     0x7ffff7ff3000    0x23000     0x1000
↪  /usr/lib/x86_64-linux-gnu/ld-2.31.so
  0x7ffff7ff3000     0x7ffff7ffb000     0x8000    0x24000
↪  /usr/lib/x86_64-linux-gnu/ld-2.31.so
  0x7ffff7ffc000     0x7ffff7ffd000     0x1000    0x2c000
↪  /usr/lib/x86_64-linux-gnu/ld-2.31.so
  0x7ffff7ffd000     0x7ffff7ffe000     0x1000    0x2d000
↪  /usr/lib/x86_64-linux-gnu/ld-2.31.so
  0x7ffff7ffe000     0x7ffff7fff000     0x1000        0x0
  0x7ffffffdc000     0x7fffffffe000    0x23000        0x0 [stack]
  0x8000001f0000     0x8000001f4000     0x4000        0x0 [vvar]
  0x8000001f4000     0x8000001f6000     0x2000        0x0 [vdso]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0 [vsyscall]
```

When we look at the enclave stream (sgxs):

```
azureuser@blindai-preview:~/enclave-playground$ sgxs-info info
↪  target/x86_64-fortanix-unknown-sgx/debug/enclave-playground.sgxs
  0-   efff Reg  r--  (data) meas=all
  f000-  33fff Reg  r-x  (data) meas=all
  34000-  36fff Reg  rw-  (data) meas=all
  37000-  37fff Reg  rw- (empty) meas=all
  38000-2037fff Reg  rw- (empty) meas=none
  2038000-2047fff (unmapped)
  2048000-2067fff Reg  rw- (empty) meas=none
```

```
  2068000-2068fff Reg  rw-  (data) meas=all
  2069000-2069fff Tcs  ---  (data) meas=all [oentry=0xfea8, ossa=0x206a000,
↪  nssa=1]
  206a000-206afff Reg  rw- (empty) meas=none
  206b000-207afff (unmapped)
  207b000-209afff Reg  rw- (empty) meas=none
  209b000-209bfff Reg  rw-  (data) meas=all
  209c000-209cfff Tcs  ---  (data) meas=all [oentry=0xfea8, ossa=0x209d000,
↪  nssa=1]
  209d000-209dfff Reg  rw- (empty) meas=none
  209e000-20adfff (unmapped)
  20ae000-20cdfff Reg  rw- (empty) meas=none
  20ce000-20cefff Reg  rw-  (data) meas=all
  20cf000-20cffff Tcs  ---  (data) meas=all [oentry=0xfea8, ossa=0x20d0000,
↪  nssa=1]
  20d0000-20d0fff Reg  rw- (empty) meas=none
  20d1000-20e0fff (unmapped)
  20e1000-2100fff Reg  rw- (empty) meas=none
  2101000-2101fff Reg  rw-  (data) meas=all
  2102000-2102fff Tcs  ---  (data) meas=all [oentry=0xfea8, ossa=0x2103000,
↪  nssa=1]
  2103000-2103fff Reg  rw- (empty) meas=none
  2104000-3ffffff (unmapped)
```

We notice a clear mapping between the process in memory and the SGXS stream.

```
  0x7ffff0000000     0x7ffff000f000     0xf000        0x0 /dev/sgx_enclave
  0x7ffff000f000     0x7ffff0034000    0x25000        0x0 /dev/sgx_enclave
  0x7ffff0034000     0x7ffff2038000  0x2004000        0x0 /dev/sgx_enclave
  0x7ffff2038000     0x7ffff2048000    0x10000  0x2771000 /dev/zero (deleted)
  0x7ffff2048000     0x7ffff206b000    0x23000        0x0 /dev/sgx_enclave
  0x7ffff206b000     0x7ffff207b000    0x10000  0x27a4000 /dev/zero (deleted)
  0x7ffff207b000     0x7ffff209e000    0x23000        0x0 /dev/sgx_enclave
  0x7ffff209e000     0x7ffff20ae000    0x10000  0x27d7000 /dev/zero (deleted)
  0x7ffff20ae000     0x7ffff20d1000    0x23000        0x0 /dev/sgx_enclave
  0x7ffff20d1000     0x7ffff20e1000    0x10000  0x280a000 /dev/zero (deleted)
  0x7ffff20e1000     0x7ffff2104000    0x23000        0x0 /dev/sgx_enclave
  0x7ffff2104000     0x7ffff4000000  0x1efc000  0x283d000 /dev/zero (deleted)
```

A.5 Default parameters of enclave builder lead to a debuggable enclave

The default setup of `EnclaveBuilder` keeps the `DEBUG` attributes in the enclave.

```
// Running the enclave
let file = parse_args().unwrap();
let aesm_client = AesmClient::new();
let mut device = IsgxDevice::new()
```

```
.unwrap()
.einittoken_provider(aesm_client)
.build();
let enclave_builder = EnclaveBuilder::new(file.as_ref());

let enclave = enclave_builder.build(&mut device).unwrap();
```

The `build` function calls the `load` function

```
pub fn build<T: Load>(mut self, loader: &mut T) -> Result<Command, Error> {
    let args = self.cmd_args.take().unwrap_or_default();
    let c = self.usercall_ext.take();
    self.load(loader)
    .map(|(t, a, s, fp)| Command::internal_new(t, a, s, c, fp, args))
  }
```

Which calls the `generate_dummy_signature` function if no signature has been found (which is the case for this assessment).

```
fn load<T: Load>(
mut self,
loader: &mut T,
) -> Result<(Vec<ErasedTcs>, *mut c_void, usize, bool), Error> {
    let signature = match self.signature {
        Some(sig) => sig,
        None => self
        .generate_dummy_signature()
        .context("While generating dummy signature")?,
      };
    let attributes = self.attributes.unwrap_or(signature.attributes);
    let miscselect = self.miscselect.unwrap_or(signature.miscselect);
    let mapping = loader.load(&mut self.enclave, &signature, attributes,
↪   miscselect)?;
    let forward_panics = self.forward_panics;
    if mapping.tcss.is_empty() {
        unimplemented!()
      }
    Ok((
    mapping.tcss.into_iter().map(ErasedTcs::new).collect(),
    mapping.info.address(),
    mapping.info.size(),
    forward_panics,
    ))
  }
```

The `generate_dummy_signature` use a default attribute with the `DEBUG` flag set.

```
fn generate_dummy_signature(&mut self) -> Result<Sigstruct, Error> {
    fn xgetbv0() -> u64 {
        unsafe { arch::x86_64::_xgetbv(0) }
```

```
      }

    let mut enclave = self.enclave.try_clone().unwrap();
    let hash = match self.hash_enclave.take() {
        Some(f) => f(&mut enclave)?,
        None => return Err(format_err!("either compile with default features or
↪  use with_dummy_signature_signer()"))
      };
    let mut signer = Signer::new(hash);

    let attributes = self.attributes.unwrap_or_else(|| Attributes {
        flags: AttributesFlags::DEBUG | AttributesFlags::MODE64BIT,
        xfrm: xgetbv0(),
      });
    signer
    .attributes_flags(attributes.flags, !0)
    .attributes_xfrm(attributes.xfrm, !0);

    if let Some(miscselect) = self.miscselect {
        signer.miscselect(miscselect, !0);
      }

    match self.load_and_sign.take() {
        Some(f) => f(signer),
        None => Err(format_err!("either compile with default features or use
↪  with_dummy_signature_signer()"))
      }
  }
```

This means that the enclave can be debugged. It's quite straightforward to confirm that as shown below.

```
azureuser@blindai-preview:~/blindai-preview$ sudo gdb -p 482872
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
Attaching to process 482872
  [New LWP 482873]
  [New LWP 482897]
  [New LWP 482898]
```

```
  [New LWP 482899]
  [New LWP 482900]
  [Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
0x00007f7bf041146e in epoll_wait (epfd=8, events=0x561384541110, maxevents=1024,
↪  timeout=8) at ../sysdeps/unix/sysv/linux/epoll_wait.c:30
30      ../sysdeps/unix/sysv/linux/epoll_wait.c: No such file or directory.
(gdb) x/10x 0x7f7afcc82cb8
0x7f7afcc82cb8: 0x01255ab0      0x00007f7a      0x00000001      0x00000000
0x7f7afcc82cc8: 0x0127a380      0x00007f7a      0x0127af80      0x00007f7a
0x7f7afcc82cd8: 0x00000000      0x00000000
```

The fix is quite simple, one just needs to set up the correct attributes before calling the `build` function (or use a `.sig` as Fortanix EDP documentation suggests).

```
let mut enclave_builder = EnclaveBuilder::new(file.as_ref());

let attributes = Attributes {
    flags: AttributesFlags::MODE64BIT,
    xfrm: xgetbv0(),
  };

enclave_builder.attributes(attributes);

let enclave = enclave_builder.build(&mut device).unwrap();
```

By doing so, the auditors can attach a debugger. However, it is not possible to access the memory of the enclave (which is expected).

```
(gdb) x/10xg 0x7f7afcc82cb8
0x7f7afcc82cb8: Cannot access memory at address 0x7f7afcc82cb8
```

A.6 Output of `cargo geiger`

```
$ CARGO_BUILD_TARGET="x86_64-fortanix-unknown-sgx" cargo geiger --output-format
↪  Ascii

Metric output format: x/y
    x = unsafe code used by the build
    y = total unsafe code found in the crate

Symbols:
    :) = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ?  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    !  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency
```

```
19/567     7981/32405   26/516 5/62    101/720  !  blindai_server 0.0.1
15/18      453/460      3/3    0/0     12/12    !  |-- anyhow 1.0.68
24/24      690/744      11/13  1/1     16/20    !  |-- bytes 1.3.0
0/0        5/5          0/0    0/0     0/0      !  |   `-- serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ?  |       `-- serde_derive
↪  1.0.152
0/0        15/15        0/0    0/0     3/3      !  |           |-- proc-macro2
↪  1.0.50
0/0        4/4          0/0    0/0     0/0      !  |           |   `--
↪  unicode-ident 1.0.6
0/0        0/0          0/0    0/0     0/0      ?  |           |-- quote 1.0.23
0/0        15/15        0/0    0/0     3/3      !  |           |   `-- proc-macro2
↪  1.0.50
0/0        69/69        3/3    0/0     2/2      !  |           `-- syn 1.0.107
0/0        15/15        0/0    0/0     3/3      !  |               |-- proc-macro2
↪  1.0.50
0/0        0/0          0/0    0/0     0/0      ?  |               |-- quote
↪  1.0.23
0/0        4/4          0/0    0/0     0/0      !  |               `--
↪  unicode-ident 1.0.6
0/0        0/0          0/0    0/0     0/0      :) |-- digest 0.10.6
0/0        16/16        0/0    0/0     0/0      !  |   |-- block-buffer 0.10.3
1/1        285/285      20/20  8/8     5/5      !  |   |   `-- generic-array
↪  0.14.6
0/0        5/5          0/0    0/0     0/0      !  |   |       |-- serde 1.0.152
0/0        0/0          0/0    0/0     0/0      :) |   |       |-- typenum 1.16.0
1/1        22/22        0/0    0/0     0/0      !  |   |       `-- zeroize 1.5.7
0/0        5/5          0/0    0/0     0/0      !  |   |           `-- serde
↪  1.0.152
0/0        0/0          0/0    0/0     0/0      :) |   `-- crypto-common 0.1.6
1/1        285/285      20/20  8/8     5/5      !  |       |-- generic-array
↪  0.14.6
0/0        2/2          0/0    0/0     0/0      !  |       |-- rand_core 0.6.4
2/4        42/175       0/1    0/0     0/3      !  |       |   |-- getrandom 0.2.8
0/0        0/0          0/0    0/0     0/0      ?  |       |   |   |-- cfg-if
↪  1.0.0
0/24       0/444        0/2    0/0     0/45     ?  |       |   |   `-- libc
↪  0.2.139
0/0        5/5          0/0    0/0     0/0      !  |       |   `-- serde 1.0.152
0/0        0/0          0/0    0/0     0/0      :) |       `-- typenum 1.16.0
0/0        0/0          0/0    0/0     0/0      ?  |-- env_logger 0.10.0
1/1        16/18        1/1    0/0     0/0      !  |   |-- log 0.4.17
0/0        0/0          0/0    0/0     0/0      ?  |   |   |-- cfg-if 1.0.0
0/0        5/5          0/0    0/0     0/0      !  |   |   `-- serde 1.0.152
0/0        34/34        1/2    0/0     2/2      !  |   `-- regex 1.7.1
19/19      678/678      0/0    0/0     22/22    !  |       |-- aho-corasick 0.7.20
36/37      2067/2144    0/0    0/0     21/21    !  |       |   `-- memchr 2.5.0
0/24       0/444        0/2    0/0     0/45     ?  |       |       `-- libc
↪  0.2.139
36/37      2067/2144    0/0    0/0     21/21    !  |       |-- memchr 2.5.0
0/0        0/0          0/0    0/0     0/0      :) |       `-- regex-syntax 0.6.28
1/1        16/18        1/1    0/0     0/0      !  |-- log 0.4.17
```

```
0/0        0/0          0/0    0/0     0/0      ?  |-- num-derive 0.3.3
0/0        15/15        0/0    0/0     3/3      !  |   |-- proc-macro2 1.0.50
0/0        0/0          0/0    0/0     0/0      ?  |   |-- quote 1.0.23
0/0        69/69        3/3    0/0     2/2      !  |   `-- syn 1.0.107
0/0        6/12         0/0    0/0     0/0      !  |-- num-traits 0.2.15
0/0        8/8          0/0    0/0     0/0      !  |   `-- libm 0.2.6
0/0        32/32        0/0    0/0     0/0      !  |-- rand 0.8.5
0/24       0/444        0/2    0/0     0/45     ?  |   |-- libc 0.2.139
1/1        16/18        1/1    0/0     0/0      !  |   |-- log 0.4.17
0/0        0/0          0/0    0/0     0/0      ?  |   |-- rand_chacha 0.3.1
2/2        636/712      0/0    0/0     17/25    !  |   |   |-- ppv-lite86 0.2.17
0/0        2/2          0/0    0/0     0/0      !  |   |   |-- rand_core 0.6.4
0/0        5/5          0/0    0/0     0/0      !  |   |   `-- serde 1.0.152
0/0        2/2          0/0    0/0     0/0      !  |   |-- rand_core 0.6.4
0/0        5/5          0/0    0/0     0/0      !  |   `-- serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ?  |-- rcgen 0.10.0
1/1        471/471      13/13  5/5     1/1      !  |   |-- ring 0.16.20
0/24       0/444        0/2    0/0     0/45     ?  |   |   |-- libc 0.2.139
1/1        92/138       5/9    0/0     2/4      !  |   |   |-- once_cell 1.17.0
0/0        49/49        6/6    0/0     3/3      !  |   |   |-- spin 0.5.2
0/0        0/0          0/0    0/0     0/0      :) |   |   `-- untrusted 0.7.1
0/1        4/32         0/0    0/0     0/0      !  |   |-- time 0.3.17
0/0        7/7          0/0    0/0     0/0      !  |   |   |-- itoa 1.0.5
0/24       0/444        0/2    0/0     0/45     ?  |   |   |-- libc 0.2.139
0/0        32/32        0/0    0/0     0/0      !  |   |   |-- rand 0.8.5
0/0        5/5          0/0    0/0     0/0      !  |   |   |-- serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ?  |   |   |-- time-core 0.1.0
0/0        0/0          0/0    0/0     0/0      ?  |   |   `-- time-macros 0.2.6
0/0        0/0          0/0    0/0     0/0      ?  |   |       `-- time-core 0.1.0
0/0        0/0          0/0    0/0     0/0      :) |   |-- yasna 0.5.1
0/0        4/4          0/0    0/0     2/2      !  |   |   |-- bit-vec 0.6.3
0/0        5/5          0/0    0/0     0/0      !  |   |   |   `-- serde 1.0.152
0/0        6/11         0/0    0/0     0/0      !  |   |   |-- num-bigint 0.4.3
0/0        0/0          0/0    0/0     0/0      ?  |   |   |   |-- num-integer
↪  0.1.45
0/0        6/12         0/0    0/0     0/0      !  |   |   |   |   `-- num-traits
↪  0.2.15
0/0        6/12         0/0    0/0     0/0      !  |   |   |   |-- num-traits
↪  0.2.15
0/0        32/32        0/0    0/0     0/0      !  |   |   |   |-- rand 0.8.5
0/0        5/5          0/0    0/0     0/0      !  |   |   |   `-- serde 1.0.152
0/1        4/32         0/0    0/0     0/0      !  |   |   `-- time 0.3.17
1/1        22/22        0/0    0/0     0/0      !  |   `-- zeroize 1.5.7
1/1        471/471      13/13  5/5     1/1      !  |-- ring 0.16.20
0/0        0/0          0/0    0/0     0/0      ?  |-- rouille 3.6.1
0/0        0/0          0/0    0/0     0/0      :) |   |-- base64 0.13.1
0/0        0/48         2/2    0/0     0/0      !  |   |-- chrono 0.4.23
0/2        1/147        0/0    0/0     0/1      !  |   |   |-- iana-time-zone
↪  0.1.53
0/0        0/0          0/0    0/0     0/0      ?  |   |   |-- num-integer 0.1.45
0/0        6/12         0/0    0/0     0/0      !  |   |   |-- num-traits 0.2.15
0/0        5/5          0/0    0/0     0/0      !  |   |   `-- serde 1.0.152
```

```
0/0         0/0          0/0    0/0     0/0      ?  |   |-- multipart 0.18.0
0/0         80/100       0/0    0/0     6/8      !  |   |   |-- buf_redux 0.8.4
36/37       2067/2144    0/0    0/0     21/21    !  |   |   |   |-- memchr 2.5.0
0/0         15/15        0/0    0/0     0/0      !  |   |   |   `-- safemem 0.3.3
14/14       273/273      0/0    0/0     4/4      !  |   |   |-- httparse 1.8.0
0/0         7/7          1/1    0/0     0/0      !  |   |   |-- lazy_static 1.4.0
0/0         49/49        6/6    0/0     3/3      !  |   |   |   `-- spin 0.5.2
1/1         16/18        1/1    0/0     0/0      !  |   |   |-- log 0.4.17
0/0         0/2          0/0    0/0     0/0      ?  |   |   |-- mime 0.3.16
0/0         0/0          0/0    0/0     0/0      ?  |   |   |-- mime_guess 2.0.4
0/0         0/2          0/0    0/0     0/0      ?  |   |   |   |-- mime 0.3.16
0/0         0/0          0/0    0/0     0/0      ?  |   |   |   `-- unicase 2.6.0
0/0         0/0          0/0    0/0     0/0      ?  |   |   |-- quick-error 1.2.3
0/0         32/32        0/0    0/0     0/0      !  |   |   |-- rand 0.8.5
0/0         15/15        0/0    0/0     0/0      !  |   |   |-- safemem 0.3.3
0/0         0/71         0/0    0/0     0/0      ?  |   |   |-- tempfile 3.3.0
0/0         0/0          0/0    0/0     0/0      ?  |   |   |   |-- cfg-if 1.0.0
0/0         0/0          0/0    0/0     0/0      :) |   |   |   |-- fastrand 1.8.0
0/24        0/444        0/2    0/0     0/45     ?  |   |   |   |-- libc 0.2.139
0/0         0/79         0/0    0/0     0/0      ?  |   |   |   `-- remove_dir_all
↪  0.5.3
0/3         0/162        0/0    0/0     0/0      :) |   |   `-- twoway 0.1.8
36/37       2067/2144    0/0    0/0     21/21    !  |   |       `-- memchr 2.5.0
0/0         65/72        0/0    0/0     0/0      !  |   |-- num_cpus 1.15.0
0/24        0/444        0/2    0/0     0/45     ?  |   |   `-- libc 0.2.139
0/0         3/3          0/0    0/0     0/0      !  |   |-- percent-encoding 2.2.0
0/0         32/32        0/0    0/0     0/0      !  |   |-- rand 0.8.5
0/0         5/5          0/0    0/0     0/0      !  |   |-- serde 1.0.152
0/0         0/0          0/0    0/0     0/0      ?  |   |-- serde_derive 1.0.152
0/0         4/7          0/0    0/0     0/0      !  |   |-- serde_json 1.0.91
0/0         7/7          0/0    0/0     0/0      !  |   |   |-- itoa 1.0.5
7/9         579/715      0/0    0/0     2/2      !  |   |   |-- ryu 1.0.12
0/0         5/5          0/0    0/0     0/0      !  |   |   `-- serde 1.0.152
1/1         83/83        0/0    0/0     0/0      !  |   |-- sha1 0.10.5
0/0         0/0          0/0    0/0     0/0      ?  |   |   |-- cfg-if 1.0.0
0/1         0/14         0/0    0/0     0/0      ?  |   |   |-- cpufeatures 0.2.5
0/0         0/0          0/0    0/0     0/0      :) |   |   `-- digest 0.10.6
0/0         0/0          0/0    0/0     0/0      ?  |   |-- threadpool 1.8.1
0/0         65/72        0/0    0/0     0/0      !  |   |   `-- num_cpus 1.15.0
0/1         4/32         0/0    0/0     0/0      !  |   |-- time 0.3.17
0/0         0/0          0/0    0/0     0/0      :) |   |-- tiny_http 0.12.0
0/0         180/180      0/0    0/0     28/28    !  |   |   |-- ascii 1.1.0
0/0         5/5          0/0    0/0     0/0      !  |   |   |   `-- serde 1.0.152
0/0         0/0          0/0    0/0     0/0      ?  |   |   |-- chunked_transfer
↪  1.4.1
0/0         0/0          0/0    0/0     0/0      :) |   |   |-- httpdate 1.0.2
1/1         16/18        1/1    0/0     0/0      !  |   |   |-- log 0.4.17
0/0         0/5          0/0    0/0     0/0      :) |   |   |-- rustls 0.20.8
1/1         16/18        1/1    0/0     0/0      !  |   |   |   |-- log 0.4.17
1/1         471/471      13/13  5/5     1/1      !  |   |   |   |-- ring 0.16.20
0/0         0/0          0/0    0/0     0/0      :) |   |   |   |-- sct 0.7.0
```

```
1/1        471/471      13/13  5/5     1/1      !  |   |   |   |   |-- ring
↪  0.16.20
0/0        0/0          0/0    0/0     0/0      :) |   |   |   |   `-- untrusted
↪  0.7.1
0/0        0/0          0/0    0/0     0/0      ?  |   |   |   `-- webpki 0.22.0
1/1        471/471      13/13  5/5     1/1      !  |   |   |       |-- ring
↪  0.16.20
0/0        0/0          0/0    0/0     0/0      :) |   |   |       `-- untrusted
↪  0.7.1
0/0        0/0          0/0    0/0     0/0      :) |   |   |-- rustls-pemfile
↪  0.2.1
0/0        0/0          0/0    0/0     0/0      :) |   |   |   `-- base64 0.13.1
1/1        22/22        0/0    0/0     0/0      !  |   |   `-- zeroize 1.5.7
0/0        0/0          0/0    0/0     0/0      ?  |   `-- url 2.3.1
0/0        2/2          0/0    0/0     0/0      !  |       |-- form_urlencoded
↪  1.1.0
0/0        3/3          0/0    0/0     0/0      !  |       |   `--
↪  percent-encoding 2.2.0
0/0        0/0          0/0    0/0     0/0      ?  |       |-- idna 0.3.0
0/0        0/0          0/0    0/0     0/0      :) |       |   |-- unicode-bidi
↪  0.3.10
0/0        5/5          0/0    0/0     0/0      !  |       |   |   `-- serde
↪  1.0.152
0/0        20/20        0/0    0/0     0/0      !  |       |   `--
↪  unicode-normalization 0.1.22
0/0        0/0          0/0    0/0     0/0      :) |       |       `-- tinyvec
↪  1.6.0
0/0        5/5          0/0    0/0     0/0      !  |       |           |-- serde
↪  1.0.152
0/0        0/0          0/0    0/0     0/0      ?  |       |           `--
↪  tinyvec_macros 0.1.0
0/0        3/3          0/0    0/0     0/0      !  |       |-- percent-encoding
↪  2.2.0
0/0        5/5          0/0    0/0     0/0      !  |       `-- serde 1.0.152
0/0        5/5          0/0    0/0     0/0      !  |-- serde 1.0.152
0/0        16/16        0/0    0/0     0/0      !  |-- serde_bytes 0.11.8
0/0        5/5          0/0    0/0     0/0      !  |   `-- serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ?  |-- serde_cbor 0.11.2
10/10      239/239      0/0    0/0     0/0      !  |   |-- half 2.2.1
18/18      414/414      128/129 9/9     0/0      !  |   |   |-- bytemuck 1.13.0
0/0        6/12         0/0    0/0     0/0      !  |   |   |-- num-traits 0.2.15
0/0        5/5          0/0    0/0     0/0      !  |   |   `-- serde 1.0.152
0/0        5/5          0/0    0/0     0/0      !  |   `-- serde 1.0.152
0/0        0/0          0/0    0/0     0/0      ?  |-- serde_derive 1.0.152
0/0        4/7          0/0    0/0     0/0      !  |-- serde_json 1.0.91
0/0        51/51        0/0    0/0     0/0      !  |-- sgx-isa 0.4.0
0/0        0/0          0/0    0/0     0/0      ?  |   |-- bitflags 1.3.2
0/0        5/5          0/0    0/0     0/0      !  |   `-- serde 1.0.152
0/0        0/0          0/0    0/0     0/0      :) |-- tiny_http 0.12.0
0/0        0/0          0/0    0/0     0/0      :) |-- ureq 2.6.2
0/0        0/0          0/0    0/0     0/0      :) |   |-- base64 0.13.1
0/2        41/118       0/2    0/0     0/2      !  |   |-- flate2 1.0.25
```

```
5/6         108/156       0/0    0/0     0/0      !  |   |   |-- crc32fast 1.3.2
0/0         0/0           0/0    0/0     0/0      ?  |   |   |   `-- cfg-if 1.0.0
0/0         0/0           0/0    0/0     0/0      :) |   |   `-- miniz_oxide 0.6.2
0/0         0/0           0/0    0/0     0/0      :) |   |       `-- adler 1.0.2
1/1         16/18         1/1    0/0     0/0      !  |   |-- log 0.4.17
1/1         92/138        5/9    0/0     2/4      !  |   |-- once_cell 1.17.0
0/0         0/5           0/0    0/0     0/0      :) |   |-- rustls 0.20.8
0/0         5/5           0/0    0/0     0/0      !  |   |-- serde 1.0.152
0/0         4/7           0/0    0/0     0/0      !  |   |-- serde_json 1.0.91
0/0         0/0           0/0    0/0     0/0      ?  |   |-- url 2.3.1
0/0         0/0           0/0    0/0     0/0      ?  |   |-- webpki 0.22.0
0/0         0/0           0/0    0/0     0/0      ?  |   `-- webpki-roots 0.22.6
0/0         0/0           0/0    0/0     0/0      ?  |       `-- webpki 0.22.0
0/0         46/46         0/0    0/0     0/0      !  `-- uuid 1.2.2
2/4         42/175        0/1    0/0     0/3      !      |-- getrandom 0.2.8
0/0         32/32         0/0    0/0     0/0      !      |-- rand 0.8.5
0/0         5/5           0/0    0/0     0/0      !      `-- serde 1.0.152

177/767     15921/42041   220/723 28/85   249/935

error: Found 26 warnings
```

A.7 Output of `cargo clippy`

```
$ azureuser@blindai-preview:~/blindai-preview$ cargo clippy --no-deps --target
↪  x86_64-fortanix-unknown-sgx -- -A clippy::all -W clippy::integer_arithmetic
↪ -W clippy::string_slice -W clippy::expect_used -W clippy::fallible_impl_from -W
↪  clippy::get_unwrap -W clippy::index_refutable_slice -W
↪  clippy::indexing_slicing -W clippy::match_on_vec_items -W
↪  clippy::match_wild_err_arm -W clippy::missing_panics_doc -W clippy::panic -W
↪  clippy::panic_in_result_fn -W clippy::unreachable -W clippy::unwrap_in_result
↪ -W clippy::unwrap_used
warning: unused import: `std::fs`
  --> rouille/src/assets.rs:10:5
   |
10 | use std::fs;
   |     ^^^^^^^
   |
   = note: `#[warn(unused_imports)]` on by default

warning: unused variable: `response`
   --> rouille/src/content_encoding.rs:131:9
    |
131 | fn gzip(response: &mut Response) {}
    |         ^^^^^^^^ help: if this is intentional, prefix it with an underscore:
↪  `_response`
    |
    = note: `#[warn(unused_variables)]` on by default

warning: unused variable: `response`
```

```
   --> rouille/src/content_encoding.rs:150:11
    |
150 | fn brotli(response: &mut Response) {}
    |           ^^^^^^^^ help: if this is intentional, prefix it with an
↪  underscore: `_response`

warning: unused variable: `request`
   --> rouille/src/assets.rs:143:32
    |
143 | pub fn match_assets<P: ?Sized>(request: &Request, path: &P) -> Response
    |                                ^^^^^^^ help: if this is intentional, prefix
↪  it with an underscore: `_request`

warning: unused variable: `path`
   --> rouille/src/assets.rs:143:51
    |
143 | pub fn match_assets<P: ?Sized>(request: &Request, path: &P) -> Response
    |                                                   ^^^^ help: if this is
↪  intentional, prefix it with an underscore: `_path`

warning: `rouille` (lib) generated 5 warnings (run `cargo clippy --fix --lib -p
↪  rouille` to apply 5 suggestions)
    Checking blindai_server v0.0.1 (/home/azureuser/blindai-preview)
warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪  std::mem::size_of::<$t>());
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
201 | impl_vec_from_to_le_bytes!(u8);
    | ------------------------------ in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: requested on the command line with `-W clippy::integer-arithmetic`
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
201 | impl_vec_from_to_le_bytes!(u8);
    | ------------------------------ in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
```

```
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪   Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪   std::mem::size_of::<$t>());
    |
↪   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
202 | impl_vec_from_to_le_bytes!(u16);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪   https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪   Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
202 | impl_vec_from_to_le_bytes!(u16);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪   https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪   Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪   std::mem::size_of::<$t>());
    |
↪   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
203 | impl_vec_from_to_le_bytes!(u32);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪   https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪   Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
```

```
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
203 | impl_vec_from_to_le_bytes!(u32);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪  std::mem::size_of::<$t>());
    |
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
↪
...
204 | impl_vec_from_to_le_bytes!(u64);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
204 | impl_vec_from_to_le_bytes!(u64);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪  std::mem::size_of::<$t>());
    |
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
↪
...
205 | impl_vec_from_to_le_bytes!(i8);
    | ------------------------------- in this macro invocation
    |
```

```
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
205 | impl_vec_from_to_le_bytes!(i8);
    | ------------------------------ in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪  std::mem::size_of::<$t>());
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
206 | impl_vec_from_to_le_bytes!(i16);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
206 | impl_vec_from_to_le_bytes!(i16);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
```

```
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪  std::mem::size_of::<$t>());
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
207 | impl_vec_from_to_le_bytes!(i32);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
207 | impl_vec_from_to_le_bytes!(i32);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪  std::mem::size_of::<$t>());
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
208 | impl_vec_from_to_le_bytes!(i64);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
208 | impl_vec_from_to_le_bytes!(i64);
    | ------------------------------- in this macro invocation
    |
```

```
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪  std::mem::size_of::<$t>());
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
209 | impl_vec_from_to_le_bytes!(f32);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
209 | impl_vec_from_to_le_bytes!(f32);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
   --> src/model.rs:183:54
    |
183 |                 let mut v = Vec::<$t>::with_capacity(bytes.len() /
↪  std::mem::size_of::<$t>());
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
210 | impl_vec_from_to_le_bytes!(f64);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: integer arithmetic detected
```

```
   --> src/model.rs:184:20
    |
184 |                 if bytes.len() % std::mem::size_of::<$t>() != 0 {
    |                    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
...
210 | impl_vec_from_to_le_bytes!(f64);
    | ------------------------------- in this macro invocation
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic
    = note: this warning originates in the macro `impl_vec_from_to_le_bytes` (in
↪  Nightly builds, run with -Z macro-backtrace for more info)

warning: indexing may panic
   --> src/model.rs:322:37
    |
322 |                     node_name: Some(output_names[i].clone()),
    |                                     ^^^^^^^^^^^^^^^
    |
    = help: consider using `.get(n)` or `.get_mut(n)` instead
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#indexing_slicing
    = note: requested on the command line with `-W clippy::indexing-slicing`

warning: used unwrap or expect in a function that returns result or option
   --> src/model_store.rs:49:5
    |
49  | /     pub fn add_model(
50  | |         &self,
51  | |         model_bytes: &[u8],
52  | |         model_name: Option<String>,
...   |
110 | |         Ok((model_id, model_hash))
111 | |     }
    | |_____^
    |
    = help: unwrap and expect should not be used in a function that returns result
↪  or option
note: potential non-recoverable error(s)
   --> src/model_store.rs:63:30
    |
63  |             let mut models = self.inner.write().unwrap();
    |                              ^^^^^^^^^^^^^^^^^^^^^^^^^^^
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_in_result
    = note: requested on the command line with `-W clippy::unwrap-in-result`

warning: used `unwrap()` on a `Result` value
  --> src/model_store.rs:63:30
   |
63 |             let mut models = self.inner.write().unwrap();
   |                              ^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

```
   |
   = help: if this value is an `Err`, it will panic
   = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_used
   = note: requested on the command line with `-W clippy::unwrap-used`

warning: integer arithmetic detected
  --> src/model_store.rs:72:21
   |
72 |                     *num += 1;
   |                     ^^^^^^^^^
   |
   = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic

warning: used unwrap or expect in a function that returns result or option
   --> src/model_store.rs:113:5
    |
113 | /     pub fn use_model<U>(&self, model_id: Uuid, fun: impl
↪  Fn(&InferenceModel) -> U) -> Option<U> {
114 | |         // take a read lock
115 | |         let read_guard = self.inner.read().unwrap();
116 | |
117 | |         read_guard.models_by_id.get(&model_id).map(fun)
118 | |     }
    | |_____^
    |
    = help: unwrap and expect should not be used in a function that returns result
↪  or option
note: potential non-recoverable error(s)
   --> src/model_store.rs:115:26
    |
115 |         let read_guard = self.inner.read().unwrap();
    |                          ^^^^^^^^^^^^^^^^^^^^^^^^^^^
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_in_result

warning: used `unwrap()` on a `Result` value
   --> src/model_store.rs:115:26
    |
115 |         let read_guard = self.inner.read().unwrap();
    |                          ^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_used

warning: used unwrap or expect in a function that returns result or option
   --> src/model_store.rs:120:5
    |
120 | /     pub fn delete_model(&self, model_id: Uuid) -> Option<InferenceModel> {
121 | |         let mut write_guard = self.inner.write().unwrap();
```

```
122 | |
123 | |         let model = match write_guard.models_by_id.entry(model_id) {
...   |
139 | |         Some(model)
140 | |     }
    | |_____^
    |
    = help: unwrap and expect should not be used in a function that returns result
↪  or option
note: potential non-recoverable error(s)
   --> src/model_store.rs:121:31
    |
121 |         let mut write_guard = self.inner.write().unwrap();
    |                               ^^^^^^^^^^^^^^^^^^^^^^^^^^
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_in_result

warning: used `unwrap()` on a `Result` value
   --> src/model_store.rs:121:31
    |
121 |         let mut write_guard = self.inner.write().unwrap();
    |                               ^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_used

warning: integer arithmetic detected
   --> src/model_store.rs:133:13
    |
133 |             *i -= 1;
    |             ^^^^^^
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic

warning: used unwrap or expect in a function that returns result or option
   --> src/client_communication.rs:89:5
    |
89  | /     pub fn send_model(&self, request: &rouille::Request) ->
↪  Result<SendModelReply, Error> {
90  | |         let upload_model_body: UploadModel = {
91  | |             let mut data: Vec<u8> = vec![];
92  | |             request
...   |
130 | |         })
131 | |     }
    | |_____^
    |
    = help: unwrap and expect should not be used in a function that returns result
↪  or option
note: potential non-recoverable error(s)
```

```
   --> src/client_communication.rs:92:13
    |
92  | /             request
93  | |                 .data()
94  | |                 .expect("Could not get input")
    | |_____________________________________________^
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_in_result

warning: used `expect()` on an `Option` value
  --> src/client_communication.rs:92:13
   |
92 | /             request
93 | |                 .data()
94 | |                 .expect("Could not get input")
   | |_____________________________________________^
   |
   = help: if this value is `None`, it will panic
   = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#expect_used
   = note: requested on the command line with `-W clippy::expect-used`

warning: used unwrap or expect in a function that returns result or option
   --> src/client_communication.rs:133:5
    |
133 | /     pub fn run_model(&self, request: &rouille::Request) ->
↪  Result<RunModelReply, Error> {
134 | |         let max_input_size = self.max_input_size;
135 | |
136 | |         let mut data_stream = request.data().expect("Could not get the
↪  input");
...   |
185 | |         Ok(RunModelReply { outputs })
186 | |     }
    | |_____^
    |
    = help: unwrap and expect should not be used in a function that returns result
↪  or option
note: potential non-recoverable error(s)
   --> src/client_communication.rs:136:31
    |
136 |         let mut data_stream = request.data().expect("Could not get the
↪  input");
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_in_result

warning: used `expect()` on an `Option` value
   --> src/client_communication.rs:136:31
    |
```

```
136 |         let mut data_stream = request.data().expect("Could not get the
↪  input");
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is `None`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#expect_used

warning: integer arithmetic detected
   --> src/client_communication.rs:142:12
    |
142 |         if run_model_body.inputs.len() * size_of::<u8>() > max_input_size
    |            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic

warning: integer arithmetic detected
   --> src/client_communication.rs:143:16
    |
143 |             || run_model_body.inputs.len() * size_of::<u8>() >
↪  max_input_size
    |                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic

warning: used unwrap or expect in a function that returns result or option
   --> src/client_communication.rs:188:5
    |
188 | /     pub fn delete_model(&self, request: &rouille::Request) -> Result<()> {
189 | |         let mut data_stream = request.data().expect("Could not get the
↪  input");
190 | |         let mut data: Vec<u8> = vec![];
191 | |         data_stream.read_to_end(&mut data)?;
...   |
206 | |         Ok(())
207 | |     }
    | |_____^
    |
    = help: unwrap and expect should not be used in a function that returns result
↪  or option
note: potential non-recoverable error(s)
   --> src/client_communication.rs:189:31
    |
189 |         let mut data_stream = request.data().expect("Could not get the
↪  input");
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_in_result
```

```
warning: used `expect()` on an `Option` value
   --> src/client_communication.rs:189:31
    |
189 |         let mut data_stream = request.data().expect("Could not get the
↪  input");
    |
↪  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is `None`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#expect_used

warning: used `unwrap()` on a `Result` value
   --> src/client_communication.rs:217:17
    |
217 |                 serde_cbor::to_vec(&reply).unwrap(),
    |                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_used

warning: used `unwrap()` on a `Result` value
   --> src/client_communication.rs:221:17
    |
221 |                 serde_cbor::to_vec(&format!("{:?}", &e)).unwrap(),
    |                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_used

warning: integer arithmetic detected
   --> src/client_communication.rs:238:23
    |
238 |         let elapsed = start.elapsed().as_micros() / repeats as u128;
    |                       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#integer_arithmetic

warning: used `unwrap()` on a `Result` value
   --> src/main.rs:107:17
    |
107 |                 serde_cbor::to_vec(&format!("{:?}", &e)).unwrap(),
    |                 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_used
```

```
warning: slicing may panic
   --> src/main.rs:134:13
    |
134 | /             rouille::router!(request,
135 | |                 (GET)(/) => {
136 | |                     debug!("Requested enclave TLS certificate");
137 | |                     respond(Bytes::new(&enclave_cert_der))
...   |
149 | |                 },
150 | |             )
    | |_____________^
    |
    = help: consider using `.get(..n)`or `.get_mut(..n)` instead
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#indexing_slicing
    = note: this warning originates in the macro `rouille::router` (in Nightly
↪  builds, run with -Z macro-backtrace for more info)

warning: used `expect()` on a `Result` value
   --> src/main.rs:154:28
    |
154 |       let untrusted_server = rouille::Server::new("0.0.0.0:9923", router)
    |  ____________________________^
155 | |         .expect("Failed to start untrusted server")
    | |_________________________________________________^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#expect_used

warning: slicing may panic
   --> src/main.rs:161:9
    |
161 | /         rouille::router!(request,
162 | |             (POST) (/upload) => {
163 | |                 let reply = exchanger_temp.send_model(request);
164 | |                 exchanger_temp.respond(request, reply)
...   |
177 | |             _ => rouille::Response::empty_404()
178 | |         )
    | |_________^
    |
    = help: consider using `.get(..n)`or `.get_mut(..n)` instead
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#indexing_slicing
    = note: this warning originates in the macro `rouille::router` (in Nightly
↪  builds, run with -Z macro-backtrace for more info)

warning: used `expect()` on a `Result` value
   --> src/main.rs:183:34
    |
183 |               let trusted_server = rouille::Server::new_ssl(
```

```
    |  _____________________________________^
184 | |                 "0.0.0.0:9924",
185 | |                 router,
186 | |                 tiny_http::SslConfig::Der(tiny_http::SslConfigDer {
...   |
190 | |             )
191 | |             .expect("Failed to start trusted server");
    | |_____________________________________________________^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#expect_used

warning: used `unwrap()` on a `Result` value
   --> src/main.rs:193:13
    |
193 |             _trusted_handle.join().unwrap();
    |             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_used

warning: used `unwrap()` on a `Result` value
   --> src/main.rs:197:5
    |
197 |     _untrusted_handle.join().unwrap();
    |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: if this value is an `Err`, it will panic
    = help: for further information visit
↪  https://rust-lang.github.io/rust-clippy/master/index.html#unwrap_used

warning: `blindai_server` (bin "blindai_server") generated 47 warnings
    Finished dev [unoptimized + debuginfo] target(s) in 0.65s
```