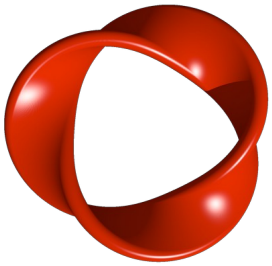

Secfault Security

Security Review Rust Intel SGX

Security Assessment

Report

for

Silence Laboratories Pte. Ltd.

Dr. Andrei Bytes
54 Flora Drive
506870 Singapore

- hereafter called "Silence Laboratories" -

Document History

Version	Author	Date	Comment
0.1	Maik Münch	2025-04-25	First Draft
0.2	Maik Münch	2025-04-29	Additions
0.3	Gregor Kopf	2025-04-30	Internal Review
0.4	Maik Münch	2025-05-15	Retest and Additions
0.5	Gregor Kopf	2025-05-16	Adjusted Company Description
0.6	Gregor Kopf	2025-06-16	Adjusted Rating of one Issue
1.0	Gregor Kopf	2025-06-20	Final Version

Table of Contents

1 Executive Summary....4
2 Overview....5
2.1 Project Description....5
2.1.1 Target Scope....5
2.1.2 Test Procedures....5
2.1.3 Coverage....6
2.2 Project Execution....6
3 Result Overview....8
4 Results....9
4.1 Out-of-bounds Array Access....9
4.2 Read of Uninitialized Memory....12
4.3 Accessing SGX-secure-vault From Multiple Operator Enclaves....14
5 Additional Observations....15
5.1 Permissions Not Cumulative....15
6 Vulnerability Rating....16
6.1 Vulnerability Types....16
6.2 Exploitability and Impact....16
7 Glossary....18

1 Executive Summary

Silence Laboratories tasked Secfault Security with the execution of a security review of parts of their "Silent Network" solution, specifically its general architecture, its usage of SGX as well as REST APIs. Further information about the project scope can be found in section 2.1.1 of this document.

The project has been executed in the time frame between 2025-04-14 and 2025-04-25 in ten person days including documentation.

The audit has been performed following a white-box approach by manually analyzing provided source code. Silence Laboratories also provided a dedicated testing environment. More detailed information about the test procedures as well as covered areas are documented in section 2.1.2 of this document.

During the security review, two memory safety issues were identified in the SGX-secret-vault. These issues might allow an attacker that has compromised an operator enclave to attack the SGX-secret-vault and potentially reveal sensitive data or execute code.

Additionally, a potential design issue was identified and documented that might allow an operator to potentially to engage in the distributed signature generation process on behalf of a different operator, effectively removing an operator from process and lowering the schemes security guarantees. Wether this issue is considered valid depends on the intended threat model and should be subject to an evaluation by Silence Laboratories.

These issues are described in more detail in section 4 of this document.

Further, Secfault Security documented a general observation made about the currently implemented permission system in section 5 of this document.

Generally, it can be positively noted that during implementation Silence Laboratories had security in mind while the code base is also well structured. The Rust code and the interfacing with foreign functions appear to be written with common security vulnerabilities and pitfalls in mind. The identified memory-related issues, however, show that there is still room for improvement. In conclusion, Secfault Security would like to point out that, overall, the solution left a positive impression.

Silence Laboratories addressed the two memory safety issues in a timely manner by ensuring the data from the client is sufficient and Secfault Security considers these issues to be fixed.

2 Overview

The following sections provide an overview of the project execution, the scope of the assessment as well as a brief summary of the test procedures applied during the engagement.

2.1 Project Description

Silence Laboratories is redefining secure computation with its Cryptographic Computing Virtual Machine (CCVM), enabling enterprises to compute on encrypted data without exposing it. At the forefront of their offerings is Silent Shard, a cutting-edge key management solution built on advanced MPC (Multi Party Computation) cryptography. Silent Shard provides unmatched speed and flexibility for companies and enterprises seeking to secure their digital assets. This powerful MPC library is already trusted and supported by numerous leading web3 companies and platforms.

The aim of this engagement was to perform a security review of the Silence Laboratories product to identify vulnerabilities or obtain sensitive information.

2.1.1 Target Scope

Silence Laboratories provided source code as well as access to a dedicated testing environment.

The following points of interest and scope were defined between Secfault Security and Silence Laboratories:

- Review of the architecture
- Review implementation and the usage of SGX
- Review RESTful APIs

Following components were explicitly defined as out of scope by Silence Laboratories:

- Cryptography parts of the solution
- Attacks against the SGX functionality itself

2.1.2 Test Procedures

For the execution of the project, Silence Laboratories provided access to relevant code repositories. Furthermore, Secfault Security was provided with a test environment to potentially analyse and verify discovered weaknesses.

The main activity of the project was a manual source code review of the solution where the code review was executed in two phases: A information gathering phase and the actual code audit.

In a first step, the documentation provided by Silence Laboratories was reviewed to collect information about the solution and to identify potentially interesting design choices that might lead to problems if not implemented correctly.

Additionally, the general architecture as well as data and information flows were inspected while paying attention to common characteristics such as multithreading. For example it was identified that external C/C++ libraries that were not considered thread-safe were used. Therefore, necessary locking mechanisms were identified to be an interesting area.

Although being mainly implemented in the programming language Rust, some parts were either implemented in C++, interacted via FFI with C/C++ libraries or used Rust's `unsafe` keyword. This observation resulted in focussing also on memory corruption issues during the actual code audit.

As a fundamental part of the solution the correct use of SGX including attestation were obligatory considered to be an interesting area.

Additionally, the WebAuthn-based Passkey implementation was also considered to be one of the major interest areas during the next analysis steps.

During the project, the review of the SGX-secret-vault revealed two memory related issues.

In general, Rust code was specifically reviewed with focus on `unsafe` blocks and the correct use of the FFI as well as logic issues that might for example stem from incomplete or wrong return value checks.

2.1.3 Coverage

The following areas were covered and prioritized during the audit, while the sole focus was put on the `el-services`:

- The SGX-secret-vault was subject to a review with focus of common memory safety issues such as buffer overflows, out-of-bounds accesses or the use of uninitialized memory.
- The remote attestation implementation was reviewed including the used caching mechanism as well as the custom certificate verifies were reviewed with focus on the correct usage of the Gramine libraries as well as potential logic issues such as caching expired certificates.
- The implemented permissions that could be associated with a key were reviewed with focus on potential short-comings such as missing commutativity. Silence Laboratories acknowledged this but replied that the current implementation is simple by design is currently on hold.
- Further, focus was put on common flaws and configuration issues of the Passkey implementation.

2.2 Project Execution

The project has been executed in the time frame from 2025-04-14 to 2025-04-25 in ten person days.

The consultants assigned to this projects were:

- Gregor Kopf
- Maik Münch

- Alexander Novikov

3 Result Overview

An overview of the project results is provided in the following table.

Description	Chapter	Type	Exploitability	Attack Impact	Status
Out-of-bounds Array Access	4.1	Code	Low	High	Fixed
Read of Uninitialized Memory	4.2	Code	Low	High	Fixed
Accessing SGX-secure-vault From Multiple Operator Enclaves	4.3	Design	Low	Medium	Acknowledged

Each identified issue is briefly described by its title, its type, its exploitability and by the impact of a successful exploitation. Technical details for the individual issues are provided in the respective sections of chapter 4 of this document. Details regarding the vulnerability rating scheme used in this document are provided in section 6.

4 Results

The issues identified during the project are described in detail in the following sections. For each finding, there is a technical description, recommended actions and - if necessary and possible - reproduction steps. For details regarding the used vulnerability rating scheme, please refer to section 6 of this document.

4.1 Out-of-bounds Array Access

Summary

Type	Location	Exploitability	Attack Impact	Status
Code	SGX-secure-vault	Low	High	Fixed

Technical Description

During manual static analysis of the SGX-secure-vault, it was identified that the function `read_from_client()` in the file `sgx-secret-vault/server/enc/tls_server_utility.cpp` is used to read a specified number of bytes (`payload_length`) from the client and stores it in a provided buffer (`payload`) and returns the number of bytes read.

However, it also accepts a zero-length as `payload_length` without errors. This semantics eventually leads to problems when attackers are able to control the `payload_length` passed to `read_from_client()`.

Such an instance was identified in the function `handle_session()` defined in the same file. The following excerpt shows the source code of said function:

```
int handle_session(SSL*& ssl_session, const kdf::SeedPtr& seed) {
  while (true) {
    uint8_t header[HEADER_SIZE] = {};

    if (read_from_client(ssl_session, header, sizeof(header)) !=
sizeof(header)) {
      t_print(TAG "Received incomplete header\n");
      return -1;
    }

    MessageType msg_type = (MessageType)header[0];
    uint16_t payload_len = ((uint16_t*)(header + 1))[0];

    t_print(TAG "Message type %d, payload_len: %d\n", msg_type, payload_len);

    if (payload_len > MAX_PAYLOAD_SIZE) {
```

```
      t_print(TAG "Invalid payload_len in header %d\n", payload_len);
      send_error_response(ssl_session, "Invalid payload_len in header");
      return -1;
    }

    switch (msg_type) {
      case MessageType::DeriveKeyReq: {
        std::vector<uint8_t> raw_value;
        raw_value.resize(payload_len);

        if (read_from_client(ssl_session, raw_value.data(), payload_len) !=
payload_len) {
          t_print(TAG "Received incomplete value\n");
          send_error_response(ssl_session, "Received incomplete value");
          return -1;
        }

        // | key_len (2 bytes) | info -> null terminated string |
        uint16_t key_len = ((uint16_t*)raw_value.data())[0];

        if (key_len == 0 || key_len > kdf::MAX_KEY_LEN) {
          t_print(TAG "Invalid keylen in request %d\n", key_len);
          send_error_response(ssl_session, "Invalid keylen in request");
          return -1;
        }

        // TODO: test it, set payload_len > than info
        raw_value[payload_len - 1] = '\0';
        std::string key_name((const char*)(raw_value.data() +
sizeof(key_len)));
...
}
```

This function first retrieves a message header comprised of a message type (`msg_type`) and a payload length (`payload_len`) and then validates if the provided payload length exceeds the given maximum payload size.

Following this, it reads `payload_len` number of bytes into a `std::vector` (`raw_value`) and finally "null-terminates" that vector at offset `payload_len - 1`.

However, as there is no check for a zero-length `payload_len` this offset might underflow, thus resulting in an out-of-bounds array write access if the provided `payload_len` was zero.

This might allow an attacker that successfully compromised an operator conclave to trigger a memory corruption that leads to undefined behaviour that might potentially be exploitable and lead to code execution.

Recommended Action

In order to mitigate this issue, it should be ensured that the provided `payload_len` conforms to all size requirements and non-conforming requests should be rejected immediately.

Reproduction Steps

This issue was identified during static code analysis and given its complex nature no Proof-of-Concept was implemented. Please refer to the details section of this issue to for reproduction.

Retest Status

This issue was addressed by ensuring that sufficient data was read and that the data sent by the client is already null-terminated.

4.2 Read of Uninitialized Memory

Summary

Type	Location	Exploitability	Attack Impact	Status
Code	SGX-secure-vault	Low	High	Fixed

Technical Description

The manual static analysis of the SGX-secure-vault revealed an issue the leads to reading and eventually using uninitialized memory.

The following excerpt of the file `sgx-secret-vault/server/enc/tls_server_utility.cpp` shows the affected source code of the function `handle_session()`:

```
int handle_session(SSL*& ssl_session, const kdf::SeedPtr& seed) {
  while (true) {
    uint8_t header[HEADER_SIZE] = {};

    if (read_from_client(ssl_session, header, sizeof(header)) !=
sizeof(header)) {
      t_print(TAG "Received incomplete header\n");
      return -1;
    }

    MessageType msg_type = (MessageType)header[0];
    uint16_t payload_len = ((uint16_t*)(header + 1))[0];

    t_print(TAG "Message type %d, payload_len: %d\n", msg_type, payload_len);

    if (payload_len > MAX_PAYLOAD_SIZE) {
      t_print(TAG "Invalid payload_len in header %d\n", payload_len);
      send_error_response(ssl_session, "Invalid payload_len in header");
      return -1;
    }

    switch (msg_type) {
      case MessageType::DeriveKeyReq: {
        std::vector<uint8_t> raw_value;
        raw_value.resize(payload_len);

        if (read_from_client(ssl_session, raw_value.data(), payload_len) !=
payload_len) {
          t_print(TAG "Received incomplete value\n");
          send_error_response(ssl_session, "Received incomplete value");
          return -1;
```

```
        }

        // | key_len (2 bytes) | info -> null terminated string |
        uint16_t key_len = ((uint16_t*)raw_value.data())[0];

        if (key_len == 0 || key_len > kdf::MAX_KEY_LEN) {
          t_print(TAG "Invalid keylen in request %d\n", key_len);
          send_error_response(ssl_session, "Invalid keylen in request");
          return -1;
        }

        // TODO: test it, set payload_len > than info
        raw_value[payload_len - 1] = '\0';
        std::string key_name((const char*)(raw_value.data() +
sizeof(key_len)));
...
}
```

After reading a message header including a length field (`payload_len`), this length field is checked to not exceed the maximum payload size (`MAX_PAYLOAD_SIZE`) and finally the given number of bytes are read from the request and get stored in a `std::vector` (`raw_value`).

Then, two bytes (`uint16_t`) are read from `raw_value` to retrieve the key length (`key_len`). However, no validation is in place to ensure that enough bytes were actually available when reading from the client, thus leading to reading potentially uninitialized memory resulting in undefined behaviour.

Eventually, the key name (`key_name`) gets initialized by reading data from `raw_value` at an offset (`sizeof(key_len)`). This also results in reading potentially uninitialized memory.

An attacker with control over an operator enclave might be able to trigger this issue by not providing enough data and potentially leak very limited information about data of previous requests. Successful exploitation is considered to be very unlikely but, nonetheless, this issue should be addressed.

Recommended Action

In order to mitigate this issue, it should be ensured that the provided `payload` actually contains enough data to be read by the implementation..

Reproduction Steps

This issue was identified during static code analysis and given its complex nature no Proof-of-Concept was implemented. Please refer to the details section of this issue to for reproduction.

Retest Status

This issue was addressed by verifying, that sufficient data was read from the client.

4.3 Accessing SGX-secure-vault From Multiple Operator Enclaves

Summary

Type	Location	Exploitability	Attack Impact	Status
Design	SGX-secure-vault	Low	Medium	Acknowledged

Technical Description

A review of the trust model in place between the SGX-secret-vault and the operator enclave revealed that access from multiple operator enclaves to a single SGX-secure-vault is considered to be correct semantically.

A malicious operator with access to the still encrypted database and the SGX-secret-vault enclave of a different operator could be able to use their own operator enclave to engage in the distributed signature generation process on behalf of a different operator (providing the correct `ORIG_ID`) effectively removing one party from the process.

Although not directly leaking any sensitive data, such a scenario would result in lowering or even defeating the security guarantees of the system in case a malicious operator would have access to multiple other operator's databases and SGX-secure-vaults.

It should be noted that since - as described above - this issue does not allow for a full compromise of the solution, and since the separation of operators is not a primary security objective, this issue has been found to have only a medium impact.

Recommended Action

Assuming that this scenario is considered to be a valid threat, it should be evaluated if a scoped trust model should be implemented by e.g., using mutual authentication between operator enclaves and SGX-secret-vaults to ensure that only the operator enclave of the operator owning the corresponding SGX-secret-vault can communicate with each other.

Reproduction Steps

Given the fact that this observation stems from a design decision no concrete reproduction steps can be provided.

5 Additional Observations

Secfault Security would like to point out a number of general observations and recommendations regarding the analyzed system in the following subsections.

5.1 Permissions Not Cumulative

While reviewing the design of the implemented permission system it was observed that permissions are applied on a per-transaction basis and are not cumulative.

To "bypass" the currently implemented permissions, a malicious user could simply issue multiple transactions where each transaction respects the permitted limit but in their entirety would exceed this limit.

It should be evaluated to implement a more complex permission system to address this issue by applying permissions cumulatively on transactions.

Silence Laboratories acknowledged this issue as the provided permission systems is currently simple by design and another permission type would be need to address the current limitations. However, this feature is currently on hold.

6 Vulnerability Rating

This section provides a description of the vulnerability rating scheme used in this document. Each finding is rated by its type and its exploitability/impact of a successful exploitation. The meaning of the individual ratings are provided in the following sub-sections.

6.1 Vulnerability Types

Vulnerabilities are rated by the types described in the following table.

Type	Description
Configuration	The finding is a configuration issue
Design	The finding is the result of a design decision
Code	The finding is caused by a coding mistake
Observation	The finding is an observation, which does not necessarily have a direct impact

6.2 Exploitability and Impact

The exploitability of a vulnerability describes the required skill level of an attacker as well as the required resources. Therefore, it provides an indication of the likelihood of exploitation.

Exploitability Rating	Description
Not Exploitable	This finding can most likely not be exploited.
Minimal	Although an attack is theoretically possible, it is extremely unlikely that an attacker will exploit the identified vulnerability.
Low	Exploiting the vulnerability requires the skill-level of an expert. An attack is possible, but difficult pre-conditions (e.g., prior identification and exploitation of other vulnerabilities) exist or the attack requires resources not available to the general public (e.g., expensive equipment). Successful exploitation indicates a dedicated, targeted attack.
Medium	The vulnerability can be exploited under certain pre-conditions (e.g., user interaction or prior authentication). Non-targeted, random attacks are possible for attackers with a medium skill level who perform such attacks on a regular basis.
High	The vulnerability can be exploited immediately without special pre-conditions, by random attackers or in an automated fashion. Only general knowledge about vulnerability exploitation is required.

The following table describes the impact rating used in this document.

Impact Rating	Description
Critical	The vulnerability is a systematic error or it permits compromising the system completely and beyond the scope of the assessment.

Impact Rating	Description
High	The vulnerability permits compromising the systems within the scope completely.
Medium	The vulnerability exceeds certain security rules, but does not lead to a full compromise (e.g., Denial of Service attacks)
Low	The vulnerability has no direct security consequences but provides information which can be used for subsequent attacks.
Informational	The observed finding does not have any direct security consequence; however, addressing the finding can lead to an increase in security or quality of the system in scope.

When rating the impact of a vulnerability, the rating is always performed based on the scope of the analysis. For example, a vulnerability with high impact typically allows an attacker to fully compromise one or all of the core security guarantees of the components in scope. Identical vulnerabilities can therefore be rated differently in different projects.

7 Glossary

Term	Definition
REST	Representational state transfer