Voltix Mobile TSS Lib Code Review

Technical Report

Thorchain

15 May 2024
Version: 1.1

Kudelski Security – Nagravision Sàrl

Corporate Headquarters
Kudelski Security – Nagravision Sàrl
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

For Public Release

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Thorchain (“the Client”) engaged Kudelski Security (“Kudelski”, “We”) to perform the Voltix Mobile TSS Lib Code Review.

The assessment was conducted remotely by the Kudelski Security Team.

The review took place between 26 April 2024 and 15 May 2024, and focused on the following objectives:

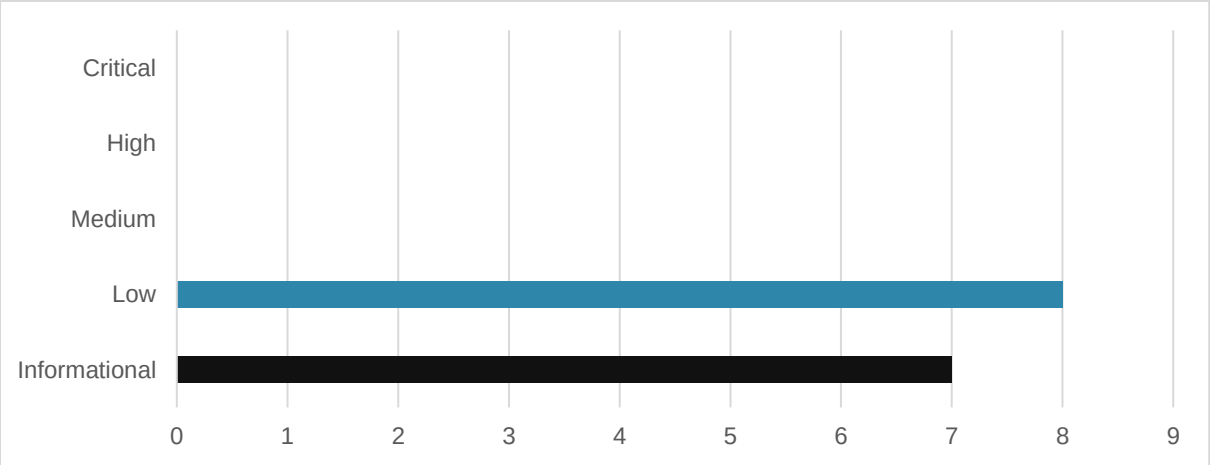
- Provide the customer with an assessment of their overall security posture and any risks that were discovered.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

Key Findings

The following are the major themes and issues identified during the testing period.

These, along with other items within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Threshold Low Limit Not Matched
- NewResharePrefix Not Validated



Findings ranked by severity

1. PROJECT SUMMARY

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

1.1 Context

The product `mobile-tss-lib`, developed by the Client, implements the threshold signature scheme library for iOS mobile devices.

1.2 Scope

The scope consisted in specific Go files and folders located at:

- Commit hash : [e3655fbff20eee2e5f247e392482503587d29d80]
- Source code repository: [https://github.com/voltix-vault/mobile-tss-lib]

The files and folders in scope are:

```
mobile-tss-lib
   ├── tss
      ├── common.go
      ├── interfaces.go
      ├── local_state.go
      ├── request_response.go
      ├── resharing.go
      ├── tss.go
```

The goal of the evaluation was to perform a security audit on the source code.

- No additional systems or resources were in scope for this assessment.
- The dependencies are out of scope of the review.
- DDOS type of attacks are out of scope.
- Collusion by dishonest party members are out of scope.
- App authentication is out of scope.

1.3 Remarks

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The code is well structured and in a good shape.
- Quick and open communication via Telegram
- The developers have made a careful and in-depth analysis of their project.
- We had regular and enriching technical exchanges on various topics.

1.4 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in ChapterVulnerability Scoring SystemVulnerability Scoring System of this document.

1.5 Follow-up

After the draft report (v1.0) was delivered, the Client addressed all findings in the following PRs:

- Audit 1 #17 (commit 06fc76f4d6d34f21fa5d1cafd1eb594d8ac4fdd7)
- Audit 2 #18 (commit 2577eb3b00d4d58a7318fa0ada726ba7965579ab)

2. TEST

2.1 Unit Tests

The unit tests were performed by the following commands.

```
$ go test ./...
ok      github.com/voltix-vault/mobile-tss-lib/tss      0.010s
```

A total of 14 test cases out of 3 test suites are written as below.

```
TestHashToInt: 3 tests
TestContains : 3 tests
TestGetDerivedPubKey: 8 tests
```

All tests passed successfully.

2.2 Coverage

The unit tests were performed by the following commands.

- Run the test with coverage profile flag:
  `go test -v -coverprofile coverage.out ./tss`
- Generate the coverage report:
  `go tool cover -html coverage.out -o coverage.html`

The total coverage rate is 8.6%; only `common.go` is `68.9%` covered, but others are `0%` covered.

3. STATIC ANALYSIS

3.1 Semgrep

Semgrep (v1.61.1) identified 4 findings and one of findings is reported in KS-THO-F-05.

3.2 codeQL

CodeQL (v2.16.6) run with the `codeql/go-queries:codeql-suites/go-security-and-quality.qls` query suite and identified 1 finding, which is reported in KS-THO-F-05.

3.3 gosec

gosec (v2.18.1) did not identify any finding.

3.4 Staticcheck

staticcheck (v0.4.7) did not identify any finding.

3.5 Errcheck

errcheck (v1.6.3) did not identify any finding.

4. TECHNICAL DETAILS OF SECURITY FINDINGS

This chapter provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the security findings.

#	SEVERITY	TITLE	STATUS
KS-THO-F-01	Low	Threshold Low Limit Not Matched	Resolved
KS-THO-F-02	Low	ECC Curve Not Validated Properly	Resolved
KS-THO-F-03	Low	GetDerivedPubKey Not Consistent	Resolved
KS-THO-F-04	Low	Weight Factor Not Consistent	Resolved
KS-THO-F-05	Low	Potential Downcasting	Resolved
KS-THO-F-06	Low	NewResharePrefix Not Validated	Resolved
KS-THO-F-07	Low	Boundary Condition Not Handled	Resolved
KS-THO-F-08	Low	DerivePath Not Validated	Resolved

Findings overview.

4.1 KS-THO-F-01 Threshold Low Limit Not Matched

Severity	Impact	Likelihood	Status
Low	Medium	Low	Resolved

Description

The function `GetThreshold` returns an error only when the input `value` is negative. If `value` is equal to `0` or `1`, it returns the corresponding threshold (`-1` or `0`) without an error. However, this is not matched with the document where the minimum number of parties should be higher than 1.

Furthermore, during the resharing process, the new parties and threshold are valid as long as no error is returned, meaning that `newPartiesCount = 1 (or 0)` and `threshold = 0 (or -1)` could be accepted although they are not valid.

4.2 KS-THO-F-02 ECC Curve Not Validated Properly

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

The function `GetHexEncodedPubKey` does not validate whether the input `pubkey` is from ECDSA or EdDSA. If the input `pubkey` is from deprecated curves, it is still encoded to the hex string without an error.

4.3 KS-THO-F-03 GetDerivedPubKey Not Consistent

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

In the function `GetDerivedPubKey` the ecc curve is set to either `secp256k1` or `Edward` curve, depending on the input `isEdDSA`. However, the function `derivingPubkeyFromPath` derives only an ECDSA key, not an EdESA key.

4.4 KS-THO-F-04 Weight Factor Not Consistent

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The weight factor is not consistent. Below is 3 but others are 2. For the resharing process, they are 1.

4.5 KS-THO-F-05 Potential Downcasting

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The output of `strconv.Atoi` may have a bigger bit size of integer than `uint32`, depending on the hardware architecture. However, it is casted to `uint32` without a proper check.

4.6 KS-THO-F-06 NewResharePrefix Not Validated

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

The new reshare prefix is not validated after it is calculated. Hence, there is a risk that the new party IDs may be manipulated in such a way that their CRC-32 checksum is same as the old one. For example, an attacker may search for the collision of the CRC-32 and assign the party IDs.

4.7 KS-THO-F-07 Boundary Condition Not Handled

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The functions `ReshareECDSA` and `ReshareEdDSA` check whether `req.LocalPartyID` belongs to the new parties or old parties. However, it might not belong to any parties. In this case, both `localPartyID` and `oldLocalPartyID` might be `nil`, but this is not explicitly handled in the functions.

4.8 KS-THO-F-08 DerivePath Not Validated

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

The function `KeysignECDSA` validates the input request by calling the function `validateKeysignRequest`. However, this function does not validate `req.DerivePath`. Hence, `req.DerivePath` can be `nil`. Note that if `GetDerivePathBytes` is called with an empty input, the function returns an empty `pathBuf` without an error.

5. OBSERVATIONS

This chapter contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

#	SEVERITY	TITLE	STATUS
KS-THO-O-01	Informational	Hard-coded Timeout	Acknowledged
KS-THO-O-02	Informational	Secure Code Practice	Resolved
KS-THO-O-03	Informational	Inefficient Coding	Resolved
KS-THO-O-04	Informational	Possible Typo	Resolved
KS-THO-O-05	Informational	CRC32 Polynomial Not Best	Acknowledged
KS-THO-O-06	Informational	ResharePrefix Not Validated	Acknowledged
KS-THO-O-07	Informational	Code Coverage Not Sufficient	Acknowledged

Observations overview.

5.1 KS-THO-O-01 Hard-coded Timeout

Description

The time out parameter for the key generation is hard-coded in the code.

5.2 KS-THO-O-02 Secure Code Practice

Description

The length of `chainCodeBuf` is not checked immediately after `chainCodeBuf` is calculated, which could lead to unnecessary code execution although the length check might be failed.

The function `getParties` is supposed to take `localPartyKey` as an input, but is sometimes abused by taking `LocalPartyID` as an input.

5.3 KS-THO-O-03 Inefficient Coding

Description

Current implementation of the function `getParties` never returns an error. However, the error is unnecessarily checked in `tss.go` (4 spots) and in `resharing.go` (4 spots).

5.4 KS-THO-O-04 Possible Typo

Description

The error message is not correct.

The comment is not completed.

5.5 KS-THO-O-05 CRC32 Polynomial Not Best

Description

According to the comments in crc32.go, the IEEE polynomial is not the best for computing the CRC-32 checksum.

5.6 KS-THO-O-06 ResharePrefix Not Validated

Description

For `KeysignECDSA` and `KeysignEdDSA`, it is not clear why `localState.ResharePrefix` is used here without validation although this is not for resharing.

5.7 KS-THO-O-07 Code Coverage Not Sufficient

Description

The total test coverage rate is only 8.6%.

- common.go: 68.9%
- others: 0%

The test coverage needs to be increased by adding more edge cases, negative tests, and regression tests. It would be nice to have fuzzing tests in addition to unit and integration tests.

6. METHODOLOGY

For this engagement, Kudelski Security used a methodology that is described at a high level in this chapter. This is broken up into the following phases.

6.1 Kickoff

The Kudelski Security Team set up a kickoff meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting, we verified the scope of the engagement and discussed the project activities.

6.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps required for gaining familiarity with the codebase and technological innovations utilized.

6.3 Review

The review phase is where most of the work on the engagement was performed. In this phase we have analyzed the project for flaws and issues that could impact the security posture. The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools was used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Review

Kudelski Security Team reviewed the code within the project utilizing an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content, to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- authentication (*e.g.* A07:2021, CWE-306)
- authorization and access control (*e.g.* A01:2021, CWE-862)
- auditing and logging (*e.g.* A09:2021)
- injection and tampering (*e.g.* A03:2021, CWE-20)
- configuration issues (*e.g.* A05:2021, CWE-798)
- logic flaws (*e.g.* A04:2021, CWE-190)
- cryptography (*e.g.* A02:2021)

These categories incorporate common weaknesses and vulnerabilities such as the OWASP Top 10 and MITRE Top 25.

Cryptography

We analyze the cryptographic primitives and components as well as their implementation. We check in particular:

- matching of the proper cryptographic primitives to the desired cryptographic functionality needed
- security level of cryptographic primitives and their respective parameters (key lengths, etc.)
- safety of the randomness generation in general as well as in the case of failure
- safety of key management
- assessment of proper security definitions and compliance to use cases
- checking for known vulnerabilities in the primitives used

6.4 Reporting

Kudelski Security delivered to The Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project.

In the report we not only point out security issues identified but also observations for improvement. The findings are categorized into several buckets, according to their overall severity: **Critical**, **High**, **Medium**, **Low**.

Observations are considered to be **Informational**. Observations can also consist of code review, issues identified during the code review that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

6.5 Verify

After the preliminary findings have been delivered, we verify the fixes applied by the Client. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase is the final report with any mitigated findings noted.

7. VULNERABILITY SCORING SYSTEM

Kudelski Security utilizes a custom approach when computing the vulnerability score, based primarily on the **Impact** of the vulnerability and **Likelihood** of an attack.

Each metric is assigned a ranking of either low, medium or high, based on the criteria defined below. The overall severity score is then computed as described in the next section.

Severity

Severity is the overall score of the finding, weakness or vulnerability as computed from Impact and Likelihood. Other factors, such as availability of tools and exploits, number of instances of the vulnerability and ease of exploitation might also be taken into account when computing the final severity score.

IMPACT / LIKELIHOOD	LOW	MEDIUM	HIGH
HIGH	MEDIUM	HIGH	HIGH
MEDIUM	LOW	MEDIUM	HIGH
LOW	LOW	LOW	MEDIUM

Compute overall severity from Impact and Likelihood. The final severity factor might vary depending on a project's specific context and risk factors.

- **Critical** The identified issue may be immediately exploitable, causing a strong and major negative impact system-wide. They should be urgently remediated or mitigated.
- **High** The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
- **Medium** The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
- **Low** The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
- **Informational** findings are best practice steps that can be used to harden the application and improve processes. Informational findings are not assigned a severity score and are classified as Informational instead.

Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

- **High** The vulnerability has a severe effect on the company and systems or has an affect within one of the primary areas of concern noted by the client.
- **Medium** It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.
- **Low** There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, difficulty of exploitation and institutional knowledge.

- **High** It is extremely likely that this vulnerability will be discovered and abused.
- **Medium** It is likely that this vulnerability will be discovered and abused by a skilled attacker.
- **Low** It is unlikely that this vulnerability will be discovered or abused when discovered.

8. REFERENCE

- Voltix overview https://docs.voltix.org/