Starke Finance Vaults Secure Code Review

Technical Report

StaRKe LLC

02 May 2025
Version: 1.1

Kudelski Security – Nagravision Sàrl

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

TABLE OF CONTENTS

EXECUTIVE SUMMARY

StaRKe LLC (“the Client”) engaged Kudelski Security (“Kudelski”, “We”) to perform the Starke Finance Vaults Secure Code Review.

The assessment was conducted remotely by the Kudelski Security Team.

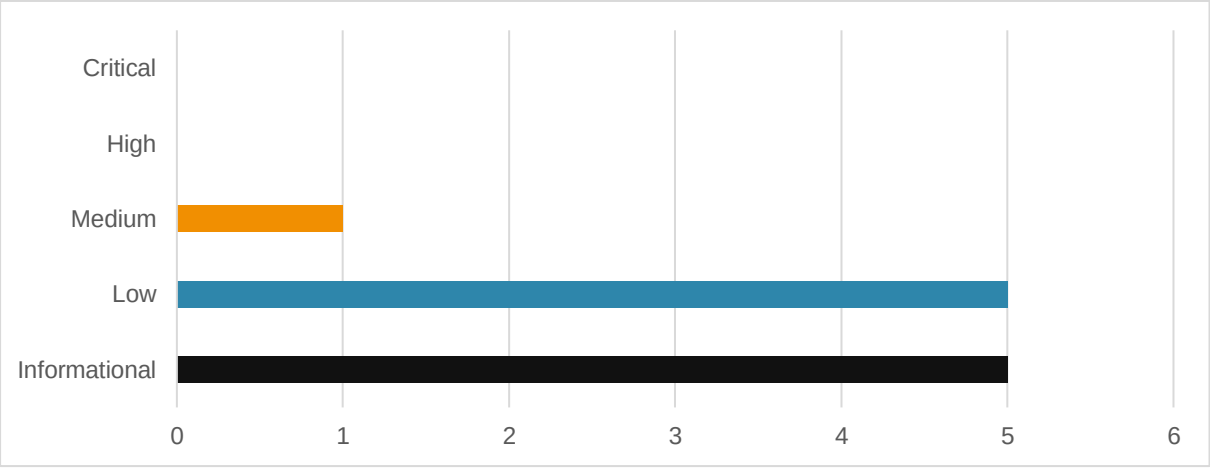
The review took place between 08 April 2025 and 16 April 2025, and focused on the following objectives:

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

- Lack of Slippage Protection in **`_swap_on_jupiter`**
- Emergency Shutdown Not Present
- Duplicate Vault Names and Metadata



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 `vaults` repository contains a secure and flexible Solana program for managing token vaults with advanced features for trustless DeFi applications.

1.2 Scope

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

- https://github.com/starke-labs/vaults

  (commit: `fe6f1b87a02e3e196d8364ecca023814d85d2373`)

- chore: refactor token program usage for vtokens and deposits
- fix: update withdrawal account chunk size from 3 to 4 to accommodate new account structure

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.
- Test codes are 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.
- Quick and open communication via Teams
- 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 Chapter Vulnerability Scoring System of this document.

1.5 Follow-up

After the initial report (V1.0) was delivered, the Client addressed or acknowledged all vulnerabilities and weaknesses in the following codebase revision:

- fix: security audit findings #4

  (commit: c745be5e4fa25289f802082565cc301eb7127656)

2. STATIC CODE ANALYSIS

2.1 Cargo Audit

`Cargo audit` (v0.21.2) identified 3 vulnerabilities and 5 warnings on dependencies. Among those, the vulnerable dependencies are reported in Chapter 3.1. The outputs of cargo audit are in the Appendix 8.

2.2 Npm audit

`npm audit` (v9.5.1) identified 16 vulnerabilities (4 moderate, 12 high) on node modules. The vulnerable dependencies are reported in Chapter 3.1. The outputs of npm audit are in the Appendix 8.

2.3 Cargo clippy

`Cargo clippy` (v0.1.83) identified 28 warnings from the code in scope, which turned out they were false alarms or not notable.

2.4 Semgrep

`Semgrep` (v1.99.0) did not identify any finding in the `program/vaults` folder.

3. 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-SRK-F-01	Medium	Lack of Slippage Protection in `_swap_on_jupiter`	Acknowledged
KS-SRK-F-02	Low	Emergency Shutdown Not Present	Resolved
KS-SRK-F-03	Low	Pyth Price Confidence Interval Not Implemented	Resolved
KS-SRK-F-04	Low	Duplicate Vault Names and Metadata	Resolved
KS-SRK-F-05	Low	Potential Underflow	Resolved
KS-SRK-F-06	Low	Zero Amount Deposit / Withdraw	Resolved

Findings overview.

3.1 KS-SRK-F-01 Lack of Slippage Protection in `_swap_on_jupiter`

Severity	Impact	Likelihood	Status
Medium	High	Low	Acknowledged

Description

The `_swap_on_jupiter` function does not validate the results of the token swap after invoking the Jupiter program. Specifically, it does not:

- Verify the post-swap balance of the `output_token_account` to ensure the correct amount of tokens was received.
- Check for slippage to ensure the received amount meets the expected minimum output.

3.2 KS-SRK-F-02 Emergency Shutdown Not Present

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

There are no emergency shutdown or halt functions in the project. In the event of a serious security issue, an emergency shutdown or a pause mechanism would be useful to halt all transactions and prevent additional damage immediately.

3.3 KS-SRK-F-03 Pyth Price Confidence Interval Not Implemented

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The Pyth network provides the price with the confidence interval. However, the confidence interval is not implemented in the `get_token_price_from_pyth_feed` function. In certain cases, it would be exposed to a price manipulation attack. Note that this is already noted in the `get_nav` function as TODO.

3.4 KS-SRK-F-04 Duplicate Vault Names and Metadata

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

Currently, managers can create vaults that share identical names, symbols, or URIs. Although each vault's PDA is unique (due to the manager's public key), this duplication can lead to user confusion or even enable phishing attempts if malicious actors create vaults with the same identifiers as legitimate ones. The code also accepts arbitrary strings for `name`, `symbol`, and `uri` without validating their length, format, or content.

3.5 KS-SRK-F-05 Potential Underflow

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The `_withdraw` function does not verify whether the input parameter `amount` is not larger than the vtoken mint supply.

3.6 KS-SRK-F-06 Zero Amount Deposit / Withdraw

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

Both `deposit` and `withdraw` functions lack validation for zero amount transactions at both instruction and controller levels. This allows users to execute transactions with amount = 0, which could waste computational resources and pollute the transaction history.

4. 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-SRK-O-01	Informational	Best Secure Code Practice	Resolved
KS-SRK-O-02	Informational	Outdated Dependencies	Resolved
KS-SRK-O-03	Informational	Latency Mitigation	Informational
KS-SRK-O-04	Informational	Lack of Unit Test Vectors	Informational
KS-SRK-O-05	Informational	Missing Countermeasure Against Price Manipulation	Informational

Observations overview.

4.1 KS-SRK-O-01 Best Secure Code Practice

Description

- When the function `checked_mul` is used to calculate the amount or price of token, a conversion to `u128` might be considered to avoid a potential overflow error. Note that some function such as `calculate_vtokens_to_mint` has such conversion.
- The order of parameter is not matched with the definition of `TransferChecked` struct: from, mint, to, and authority. Although this does not cause any issue, it would be better to make it consistent to avoid any confusion.
- The comment is not matched: should it be 4?
- Since the function accepts even when the user account does not exist, the TODO comment is valid and should be implemented.
- The vault supports both SPL token and Token 2022 program. However, it is not explicitly mentioned in the README.md or in-line comments. In comparison, it is commented that vtoken supports only SPL token at the moment, for instance, `programs/vaults/src/instructions/withdraw.rs#L103`.

4.2 KS-SRK-O-02 Outdated Dependencies

Description

The `cargo audit` (v0.21.0) tool identified 1 vulnerability on dependencies as below.

Crate	Title	RUSTSEC ID	Solution
hashbrown	Borsh serialization of HashMap is non-canonical	RUSTSEC-2024-0402	Upgrade to >=0.15.1

The `npm audit` (v9.5.1) tool identified 16 vulnerabilities (4 moderate, 12 high) on node_modules as below.

Module	Title	Severity
@babel/runtime <7.26.10	Babel has inefficient RexExp complexity in generated code with .replace when transpiling named capturing groups	Moderate
axios 1.0.0 - 1.8.1	xios Requests Vulnerable To Possible SSRF and Credential Leakage via Absolute URL	High
bigint-buffer	bigint-buffer Vulnerable to Buffer Overflow via toBigIntLE() Function	High
nanoid <3.3.8	Predictable results in nanoid generation when given non-integer values	Moderate

Module	Title	Severity
serialize-javascript 6.0.0 - 6.0.1	Cross-site Scripting (XSS) in serialize-javascript	Moderate

4.3 KS-SRK-O-03 Latency Mitigation

Description

According to the Pyth network, the latency between on-chain oracles and off-chain sources should be accounted for. If adversaries see price changes a short time before the protocol does, they may be able to make profits from the latency.

4.4 KS-SRK-O-04 Lack of Unit Test Vectors

Description

According to the `cargo llvm-cov` tool (v0.6.10), the overall test coverage of code in scope reaches less than 3%. If the test coverage is too low, hidden vulnerabilities may be introduced without being detected when the code base is updated.

4.5 KS-SRK-O-05 Missing Countermeasure Against Price Manipulation

Description

The price of token is determined by the price/data from Pyth Network. However, there is no lower/upper limit of price, compared with the previous price. This is risky due to the possibility of market/oracle price manipulation by attackers. The price from oracle is abnormally high or low, leading the incorrect estimation of the price of token.

5. 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.

5.1 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.

5.2 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.

5.3 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.

Smart Contracts

We reviewed the smart contracts, checking for additional specific issues that can arise such as:

- risk of centralization
- reentrancy
- (non)-adherence to existing standards
- unsafe arithmetic operations

5.4 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.

5.5 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.

6. 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 effect 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 effect 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.

7. CONCLUSION

The objective of this code review was to evaluate the overall security of the code base and identify any vulnerabilities that would put the product at risk.

The Kudelski Security Team identified 6 security issues: 1 medium risk and 5 low risks. On average, the effort needed to mitigate these risks is estimated as `low`.

In order to mitigate the risks posed by this engagement's findings, the Kudelski Security Team recommends applying the following best practices:

- Reload the output_token_account after the swap and calculate the difference between the pre-swap and post-swap balances to determine the amount of tokens received
- Implement an emergency shutdown mechanism which could reduce the risk from a serious security breach.
- Modify the vault creation process to include additional unique seeds (e.g., vault name) or require a name registry check.

The Client addressed or acknowledged all these vulnerabilities and observations in the follow-up revision of the codebase.

Kudelski Security remains at your disposal should you have any questions or need further assistance.

Kudelski Security would like to thank StaRKe LLC for their trust, help and support over the course of this engagement and is looking forward to cooperating in the future.

8. REFERENCE

- vaults/README.md
- Token Integration with Anchor
- Token-2022 Program