Security Review Report
NM-0392 ZkLend STRK Liquid Staking

NETHERMIND
SECURITY

(Dec 15, 2024)

Contents

1 Executive Summary

This document outlines the security review conducted by Nethermind Security for the `STRK Liquid Staking` contracts in `ZkLend`. The codebase is composed of 939 lines of Cairo code. The audit was performed using (a) manual analysis of the codebase, (b) creation of test cases, and (c) simulation of the smart contract. The source code demonstrates high writing quality.

No issues were found during the Security Review.

This document is organized as follows. Section 2 presents the files in the scope. Section 3 summarizes the issues. Section 4 presents the system overview. Section 5 discusses the risk rating methodology. Section 6 details the issues. Section 7 ensures that the deployed code matches the one audited. Section 8 discusses the documentation provided by the client for this audit. Section 9 presents the compilation, tests, and automated tests. Section 10 concludes the document.

Summary of the Audit

Audit Type	Security Review
Initial Report	Dec 15, 2024
Response from Client	Regular responses during audit engagement
Final Report	Dec 15, 2024
Repository	ZkLend-STRK-Liquid-Staking
Commit (Audit)	f52394b17c41103b95ee8522376dae7a909d3642
Commit (Final)	f52394b17c41103b95ee8522376dae7a909d3642
Documentation Assessment	High
Test Suite Assessment	High

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	src/staked_token.cairo	2	0	0.0%	1	3
2	src/interfaces.cairo	5	1	20.0%	2	8
3	src/pool.cairo	2	0	0.0%	1	3
4	src/staker.cairo	2	0	0.0%	1	3
5	src/lib.cairo	5	0	0.0%	4	9
6	src/proxy.cairo	2	0	0.0%	1	3
7	src/staked_token/staked_token.cairo	61	20	32.8%	13	94
8	src/staked_token/interface.cairo	5	1	20.0%	2	8
9	src/proxy/interface.cairo	18	1	5.6%	3	22
10	src/proxy/proxy.cairo	104	22	21.2%	17	143
11	src/staker/staker.cairo	42	20	47.6%	10	72
12	src/staker/interface.cairo	10	1	10.0%	1	12
13	src/pool/pool.cairo	636	107	16.8%	123	866
14	src/pool/interface.cairo	45	11	24.4%	19	75
	Total	**939**	**184**	**19.6%**	**198**	**1321**

3 Summary of Issues

No issues were found.

4 System Overview

The system is designed to manage a `staking` and `withdrawal system`. It primarily functions to handle the staking of funds into trenches, manage proxies that control stakes, process withdrawal requests, and distribute rewards to participants. The system manages `trench balances`, `proxy actions`, and `reward distributions`. It allows participants to stake their funds into a pool, claim rewards, and make withdrawals from the system. Proxies play a significant role in managing the staked funds and ensuring the correct processing of these activities.

The system uses the concept of `trenches` to represent pools of staked funds. Users can deposit their funds into these trenches, and the system ensures that the available funds in these trenches are sufficient to fulfill withdrawal requests. The system keeps track of the trench balance to guarantee that it can meet any withdrawal demands.

When a participant requests a withdrawal, the system places the request into a `withdrawal queue`. The system processes these requests incrementally, fulfilling them based on the available trench balance. If the funds are not sufficient to completely fulfill a withdrawal request, the system will process partial withdrawals and update the request to reflect the remaining amount. This ensures that the system can always handle requests even when funds are limited.

The system tracks the `rewards` earned by the proxies and ensures that these rewards are distributed to the correct recipients. The rewards can be claimed either during proxy deactivation or as part of the regular staking and withdrawal process. The system is responsible for collecting and distributing these rewards efficiently.

Users can deposit their funds into the system, where they are pooled into trenches. Proxies are assigned to manage these trenches and the staked funds. Proxies periodically claim rewards from the staked funds and can distribute them to the participants or use them to maintain the trench balance. Users who wish to withdraw funds submit a request, which is placed in the withdrawal queue. The system processes these requests based on the available trench balance, fulfilling them fully or partially as needed. The system ensures that the correct number of proxies is available to manage the staked funds. Proxies are deactivated when they are no longer needed, and new proxies are deployed when required.

5 Risk Rating Methodology

The risk rating methodology used by Nethermind Security follows the principles established by the OWASP Foundation. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely the finding is to be uncovered and exploited by an attacker. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage, such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage, such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage, such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding, other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Severity Risk		
Impact	**High**	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Info/Best Practices	Low	Medium
	Undetermined	Undetermined	Undetermined	Undetermined
		Low	**Medium**	**High**
		Likelihood		

To address issues that do not fit a High/Medium/Low severity, Nethermind Security also uses three more finding severities: **Informational**, **Best Practices**, and **Undetermined**.

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to pass to the client formally;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

c) **Undetermined** findings are used when we cannot predict the impact or likelihood of the issue.

6 Issues

Remarks

No issues were found.

7 Deployed Code

During this audit, we also verified that the contracts `Pool`, `StakedToken`, and `Proxy` correspond to the class hash on Mainnet.

8 Documentation Evaluation

Software documentation refers to the written or visual information that describes the functionality, architecture, design, and implementation of software. It provides a comprehensive overview of the software system and helps users, developers, and stakeholders understand how the software works, how to use it, and how to maintain it. Software documentation can take different forms, such as user manuals, system manuals, technical specifications, requirements documents, design documents, and code comments. Software documentation is critical in software development, enabling effective communication between developers, testers, users, and other stakeholders. It helps to ensure that everyone involved in the development process has a shared understanding of the software system and its functionality. Moreover, software documentation can improve software maintenance by providing a clear and complete understanding of the software system, making it easier for developers to maintain, modify, and update the software over time. Smart contracts can use various types of software documentation. Some of the most common types include:

- Technical whitepaper: A technical whitepaper is a comprehensive document describing the smart contract's design and technical details. It includes information about the purpose of the contract, its architecture, its components, and how they interact with each other;
- User manual: A user manual is a document that provides information about how to use the smart contract. It includes step-by-step instructions on how to perform various tasks and explains the different features and functionalities of the contract;
- Code documentation: Code documentation is a document that provides details about the code of the smart contract. It includes information about the functions, variables, and classes used in the code, as well as explanations of how they work;
- API documentation: API documentation is a document that provides information about the API (Application Programming Interface) of the smart contract. It includes details about the methods, parameters, and responses that can be used to interact with the contract;
- Testing documentation: Testing documentation is a document that provides information about how the smart contract was tested. It includes details about the test cases that were used, the results of the tests, and any issues that were identified during testing;
- Audit documentation: Audit documentation includes reports, notes, and other materials related to the security audit of the smart contract. This type of documentation is critical in ensuring that the smart contract is secure and free from vulnerabilities.

These types of documentation are essential for smart contract development and maintenance. They help ensure that the contract is properly designed, implemented, and tested and provide a reference for developers who need to modify or maintain it in the future.

Remarks

The ZkLend team has provided adequate documentation about the protocol.
https://github.com/zkLend/strk-liquid-staking/blob/f52394b17c41103b95ee8522376dae7a909d3642/README.md.

9 Test Suite

9.1 Project Compilation

```
scarb build
   Compiling snforge_scarb_plugin v0.33.0
    Finished `release` profile [optimized] target(s) in 0.12s
   Compiling strk_liquid_staking v0.1.0 (/Users/chris/Dropbox/NM-PYTHON/report/NM0392/strk-liquid-staking/Scarb.toml)
    Finished `dev` profile target(s) in 22 seconds
```

9.2 Test Suite

```
scarb build
   Compiling snforge_scarb_plugin v0.33.0
    Finished `release` profile [optimized] target(s) in 0.12s
   Compiling strk_liquid_staking v0.1.0 (/Users/chris/Dropbox/NM-PYTHON/report/NM0392/strk-liquid-staking/Scarb.toml)
    Finished `dev` profile target(s) in 22 seconds
chris@Chriss-MacBook-Air strk-liquid-staking % scarb tests
error: no such command: `tests`
chris@Chriss-MacBook-Air strk-liquid-staking % scarb test
     Running test strk_liquid_staking (snforge test)
   Compiling snforge_scarb_plugin v0.33.0
    Finished `release` profile [optimized] target(s) in 0.09s
   Compiling test(strk_liquid_staking_unittest) strk_liquid_staking v0.1.0
    ↪  (/Users/chris/Dropbox/NM-PYTHON/report/NM0392/strk-liquid-staking/Scarb.toml)
   Compiling test(strk_liquid_staking_tests) strk_liquid_staking_tests v0.1.0
    ↪  (/Users/chris/Dropbox/NM-PYTHON/report/NM0392/strk-liquid-staking/Scarb.toml)
    Finished `dev` profile target(s) in 38 seconds


Collected 10 test(s) from strk_liquid_staking package
Running 0 test(s) from src/
Running 10 test(s) from tests/
[PASS] strk_liquid_staking_tests::forked::test_staked_token_deflation (gas: ~3924)
[PASS] strk_liquid_staking_tests::forked::test_simple_staking (gas: ~5288)
[PASS] strk_liquid_staking_tests::forked::test_fully_fulfilled_unstake (gas: ~5478)
[PASS] strk_liquid_staking_tests::forked::test_partially_fulfilled_unstake (gas: ~6804)
[PASS] strk_liquid_staking_tests::forked::test_pre_deactivation_reward_collection (gas: ~6953)
[PASS] strk_liquid_staking_tests::forked::test_proxy_exit_cancellation (gas: ~8207)
[PASS] strk_liquid_staking_tests::forked::test_withdrawal_fulfillment_with_rewards (gas: ~7284)
[PASS] strk_liquid_staking_tests::forked::test_reward_collection (gas: ~6292)
[PASS] strk_liquid_staking_tests::forked::test_reuse_proxy (gas: ~12425)
[PASS] strk_liquid_staking_tests::forked::test_unfulfilled_unstake (gas: ~13856)
Tests: 10 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out
```

10 About Nethermind

Nethermind is a Blockchain Research and Software Engineering company. Our work touches every part of the web3 ecosystem - from layer 1 and layer 2 engineering, cryptography research, and security to application-layer protocol development. We offer strategic support to our institutional and enterprise partners across the blockchain, digital assets, and DeFi sectors, guiding them through all stages of the research and development process, from initial concepts to successful implementation.

We offer security audits of projects built on EVM-compatible chains and Starknet. We are active builders of the Starknet ecosystem, delivering a node implementation, a block explorer, a Solidity-to-Cairo transpiler, and formal verification tooling. Nethermind also provides strategic support to our institutional and enterprise partners in blockchain, digital assets, and decentralized finance (DeFi). In the next paragraphs, we introduce the company in more detail.

Blockchain Security: At Nethermind, we believe security is vital to the health and longevity of the entire Web3 ecosystem. We provide security services related to Smart Contract Audits, Formal Verification, and Real-Time Monitoring. Our Security Team comprises blockchain security experts in each field, often collaborating to produce comprehensive and robust security solutions. The team has a strong academic background, can apply state-of-the-art techniques, and is experienced in analyzing cutting-edge Solidity and Cairo smart contracts, such as ArgentX and StarkGate (the bridge connecting Ethereum and StarkNet). Most team members hold a Ph.D. degree and actively participate in the research community, accounting for 240+ articles published and 1,450+ citations in Google Scholar. The security team adopts customer-oriented and interactive processes where clients are involved in all stages of the work.

Blockchain Core Development: Our core engineering team, consisting of over 20 developers, maintains, improves, and upgrades our flagship product - the Nethermind Ethereum Execution Client. The client has been successfully operating for several years, supporting both the Ethereum Mainnet and its testnets, and now accounts for nearly a quarter of all synced Mainnet nodes. Our unwavering commitment to Ethereum's growth and stability extends to sidechains and layer 2 solutions. Notably, we were the sole execution layer client to facilitate Gnosis Chain's Merge, transitioning from Aura to Proof of Stake (PoS), and we are actively developing a full-node client to bolster Starknet's decentralization efforts. Our core team equips partners with tools for seamless node set-up, using generated docker-compose scripts tailored to their chosen execution client and preferred configurations for various network types.

DevOps and Infrastructure Management: Our infrastructure team ensures our partners' systems operate securely, reliably, and efficiently. We provide infrastructure design, deployment, monitoring, maintenance, and troubleshooting support, allowing you to focus on your core business operations. Boasting extensive expertise in Blockchain as a Service, private blockchain implementations, and node management, our infrastructure and DevOps engineers are proficient with major cloud solution providers and can host applications in-house or on clients' premises. Our global in-house SRE teams offer 24/7 monitoring and alerts for both infrastructure and application levels. We manage over 5,000 public and private validators and maintain nodes on major public blockchains such as Polygon, Gnosis, Solana, Cosmos, Near, Avalanche, Polkadot, Aptos, and StarkWare L2. Sedge is an open-source tool developed by our infrastructure experts, designed to simplify the complex process of setting up a proof-of-stake (PoS) network or chain validator. Sedge generates docker-compose scripts for the entire validator set-up based on the chosen client, making the process easier and quicker while following best practices to avoid downtime and being slashed.

Cryptography Research: At Nethermind, our Cryptography Research team is dedicated to continuous internal research while fostering close collaboration with external partners. The team has expertise across a wide range of domains, including cryptography protocols, consensus design, decentralized identity, verifiable credentials, Sybil resistance, oracles, and credentials, distributed validator technology (DVT), and Zero-knowledge proofs. This diverse skill set, combined with strong collaboration between our engineering teams, enables us to deliver cutting-edge solutions to our partners and clients.

Smart Contract Development & DeFi Research: Our smart contract development and DeFi research team comprises 40+ world-class engineers who collaborate closely with partners to identify needs and work on value-adding projects. The team specializes in Solidity and Cairo development, architecture design, and DeFi solutions, including DEXs, AMMs, structured products, derivatives, and money market protocols, as well as ERC20, 721, and 1155 token design. Our research and data analytics focuses on three key areas: technical due diligence, market research, and DeFi research. Utilizing a data-driven approach, we offer in-depth insights and outlooks on various industry themes.

Our suite of L2 tooling: Warp is Starknet's approach to EVM compatibility. It allows developers to take their Solidity smart contracts and transpile them to Cairo, Starknet's smart contract language. In the short time since its inception, the project has accomplished many achievements, including successfully transpiling Uniswap v3 onto Starknet using Warp.

- **Voyager** is a user-friendly Starknet block explorer that offers comprehensive insights into the Starknet network. With its intuitive interface and powerful features, Voyager allows users to easily search for and examine transactions, addresses, and contract details. As an essential tool for navigating the Starknet ecosystem, Voyager is the go-to solution for users seeking in-depth information and analysis;
- **Horus** is an open-source formal verification tool for StarkNet smart contracts. It simplifies the process of formally verifying Starknet smart contracts, allowing developers to express various assertions about the behavior of their code using a simple assertion language;
- **Juno** is a full-node client implementation for Starknet, drawing on the expertise gained from developing the Nethermind Client. Written in Golang and open-sourced from the outset, Juno verifies the validity of the data received from Starknet by comparing it to proofs retrieved from Ethereum, thus maintaining the integrity and security of the entire ecosystem.

Learn more about us at nethermind.io.

General Advisory to Clients

As auditors, we recommend that any changes or updates made to the audited codebase undergo a re-audit or security review to address potential vulnerabilities or risks introduced by the modifications. By conducting a re-audit or security review of the modified codebase, you can significantly enhance the overall security of your system and reduce the likelihood of exploitation. However, we do not possess the authority or right to impose obligations or restrictions on our clients regarding codebase updates, modifications, or subsequent audits. Accordingly, the decision to seek a re-audit or security review lies solely with you.

Disclaimer