Security Review Report
NM-0333 Spiko

(Nov 6, 2024)

Contents

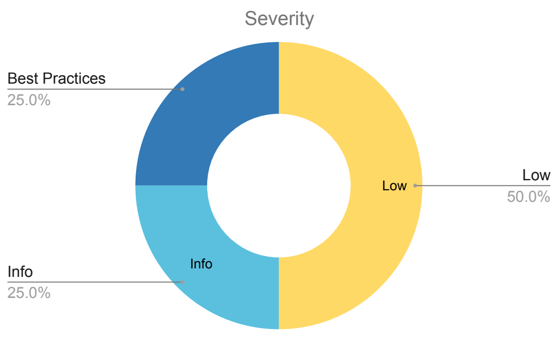
1 Executive Summary

This document presents the security review performed by Nethermind Security for Spiko smart contracts. The Spiko team has leveraged blockchain technology to issue and distribute securities. As such, they designed a protocol for managing tokenized shares in a money market fund. The security review was exclusively on their Cairo smart contracts to be deployed on Starknet.

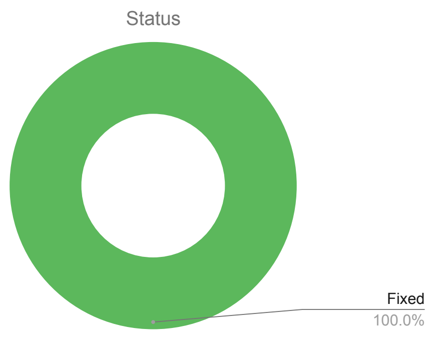
This security review focuses exclusively on the smart contracts listed in Section 2 (*Audited Files*).

The audit was performed using (a) manual analysis of the codebase and (b) creation of test cases. **Along this document, we report** 4 points of attention, where two are classified as `Low` and two are classified as `Informational` or `Best Practice`. The issues are summarized in Fig. 1.

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 discusses the documentation provided by the client for this audit. Section 8 presents the compilation, tests, and automated tests. Section 9 concludes the document.



(a)



(b)

Fig. 1: Distribution of issues: Critical (0), **High** (0), **Medium** (0), **Low** (2), **Undetermined** (0), **Informational** (1), **Best Practices** (1).
Distribution of status: Fixed (4), **Acknowledged** (0), **Mitigated** (0), **Unresolved** (0)

Summary of the Audit

Audit Type	Security Review
Initial Report	Oct 25, 2024
Response from Client	Regular responses during audit engagement
Final Report	Nov 6, 2024
Repository	starknet-contracts
Commit (Audit)	ac79ee99853dd58a77b6b1ac278c4dd4996b9337
Commit (Final)	ecb9ccb42cb85dd3c0051458da03c8db1e74c6cc
Documentation Assessment	Medium
Test Suite Assessment	Medium

2 Audited Files

	Contract	LoC	Comments	Ratio	Blank	Total
1	redemption.cairo	199	27	13.6%	22	248
2	lib.cairo	163	28	17.2%	26	217
3	permission_manager.cairo	45	15	33.3%	10	70
4	roles.cairo	6	0	0.0%	0	6
	Total	**413**	**70**	**16.9%**	**58**	**541**

3 Summary of Issues

	Finding	Severity	Update
1	Transfers of shares should be pausable	Low	Fixed
2	Users can renounce their `WHITELISTED_ROLE` , violating key protocol invariants	Low	Fixed
3	Zero amount redemption should be forbidden	Info	Fixed
4	Implement two-step transfer of ownership	Best Practices	Fixed

4 System Overview

The **Spiko** protocol introduces a Cairo-based implementation of their original Solidity contracts, designed for managing tokenized shares in a money market fund. Spiko's system isolates all investment activities outside these contracts, while minting, burning, and transferring shares are managed within the protocol, granting investors control over their fund shares.

The audit examined three primary upgradable contracts, which are described in detail in the following sections. Together, these contracts allow secure management of the fund's shares.

4.1 Permission Manager

The Permission Manager contract oversees the entire management system of the roles. The protocol enforces a role-based access control mechanism, where each role grants specific privileges. Spiko uses six custom roles and one default administrative role in this contract. Each custom role corresponds to a unique functionality:

- `WHITELISTER_ROLE`: Manages the whitelist, adding or removing users from the `WHITELISTED_ROLE`.
- `WHITELISTED_ROLE`: Authorizes users to hold and manage shares.
- `MINTER_ROLE`: Grants the ability to mint shares for whitelisted users.
- `BURNER_ROLE`: Enables the burning of shares, initially assigned to the `Redemption` contract.
- `PAUSER_ROLE`: Allows pausing the `Token` contract, preventing any new mints or burns.
- `REDEMPTION_EXECUTOR_ROLE`: Manages the redemption process, with privileges to execute or cancel redemption requests.
- `DEFAULT_ADMIN_ROLE`: Default admin role, which can grant all of the roles and manage the upgradability of the contract.

4.2 Token Contract and Redemption Contract

The Token contract functions as an ERC20 token representing shares in the money market fund. Only whitelisted users can receive these shares, which are minted by entities with the `MINTER_ROLE`. Becoming a whitelisted user requires passing a KYC process, supporting the protocol's centralized design.

After users complete Spiko's KYC process and any external asset transfers, they are whitelisted, and their shares are minted. Once shares are issued, users can manage them, including transfers and redemptions. Transfers can only occur between whitelisted users, and each transfer represents a legal event that requires user action outside the blockchain.

When users wish to redeem their shares, they initiate the process by calling the `redeem(...)` function within the `Token` contract, which is reflected in the `Redemption` contract where all redemption requests are managed:

```
fn redeem(ref self: ContractState, amount: u256, salt: felt252)
```

This function transfers shares to the `Redemption` contract, where the `on_redeem(...)` function creates a redemption request:

```
fn on_redeem(
    ref self: ContractState,
    token: ContractAddress,
    from: ContractAddress,
    amount: u256,
    salt: felt252
)
```

The redemption request is stored in a mapping, with the request's hash as the key and its status as the value. This status mechanism helps protect against replay attacks.

Once the request is created, an off-chain process is initiated, and the status of the redemption request is updated accordingly in the `Redemption` contract. Only an entity with the `REDEMPTION_EXECUTOR_ROLE` can update the request status by using one of two functions:

```
fn execute_redemption(
    ref self: ContractState,
    token: ContractAddress,
    from: ContractAddress,
    amount: u256,
    salt: felt252
)

fn cancel_redemption(
    ref self: ContractState,
    token: ContractAddress,
    from: ContractAddress,
    amount: u256,
    salt: felt252
)
```

The `execute_redemption(...)` function finalizes the redemption process by burning the shares, as the `Redemption` contract holds the `BURNER_ROLE`. This event also corresponds to the off-chain transfer of assets back to the user. Conversely, the `cancel_redemption(...)` function allows the protocol or the user to cancel the redemption request, transferring.

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

6.1 [Low] Transfers of shares should be pausable

File(s): `lib.cairo`

Description: The `Token` contract implements the `PausableComponent`, which allows the protocol to pause key operations such as minting and burning. These actions are tied to legal events that occur within the Spiko protocol, ensuring that certain operations align with off-chain legal processes. However, while minting and burning can be paused, share transfers are not currently restricted during a pause.

This issue arises because the `before_update(...)` hook function does not check the paused state of the protocol. As a result, even when the protocol is paused, users can transfer their shares. This creates an inconsistency, as share transfers may trigger legal events without any way to pause them in sync with the protocol's pause state.

Recommendation(s): Consider adding a check for the paused state in the `before_update(...)` hook function.

Status: Fixed

Update from client: c1beaa92950af6aa4320a14fb765497b087ee640

6.2 [Low] Users can renounce their `WHITELISTED_ROLE`, violating key protocol invariants

File(s): `permission_manager.cairo`

Description: Only whitelisted users are allowed to hold shares represented by the `Token` contract. This whitelist is enforced by the `before_update(...)` hook, ensuring that minting and transferring of shares are restricted to users with the `WHITELISTED_ROLE`. The `WHITELISTED_ROLE` is managed by the `PermissionManager` contract, which uses OpenZeppelin's `AccessControlComponent`. The role is assigned by users with the `WHITELISTER_ROLE`.

However, users can renounce their `WHITELISTED_ROLE` using the public `renounce_role(...)` function. This can lead to two potential issues:

1. A non-whitelisted user may end up holding shares, violating a key protocol invariant that only whitelisted users can own shares;
2. The `REDEMPTION_EXECUTOR_ROLE` cannot cancel a redemption request if the user is no longer whitelisted, which may disrupt the redemption process;

These issues can be avoided by preventing users from renouncing their `WHITELISTED_ROLE`.

Recommendation(s): Consider disallowing users with the `WHITELISTED_ROLE` from renouncing their role.

Status: Fixed

Update from client: ecb9ccb42cb85dd3c0051458da03c8db1e74c6cc

6.3 [Info] Zero amount redemption should be forbidden

File(s): `lib.cairo`

Description: The redemption process for shares occurs in two steps. First, the user calls `redeem(...)`, transferring the specified amount of shares to the `Redemption` contract. Then, a user with the `REDEMPTION_EXECUTOR_ROLE` either executes or cancels the redemption request.

However, users can currently initiate a redemption for zero shares, which leads to unnecessary gas costs for the protocol, as the request still requires execution or cancellation (Note: However, the protocol can ignore such a request). Additionally, the Spiko documentation specifies a minimum redemption amount of $1. To align with this requirement, once the Oracle is implemented, it would be beneficial to enforce a minimum share redemption threshold at the smart contract level.

Recommendation(s): Consider prohibiting redemptions for zero shares to prevent unnecessary requests.

Status: Fixed

Update from client: 538d80942b3131959f8d43a066ab92b7ff54206a

6.4 [Best Practices] Implement two-step transfer of ownership

File(s): `lib.cairo`, `redemption.cairo`

Description: The `Token` and `Redemption` contracts use the `OwnableMixinImpl` component from OpenZeppelin, which handles access control for the contract owner. Given that the protocol is highly centralized, additional safeguards should be in place to prevent accidental or unauthorized transfers of ownership to unintended addresses.

A widely recommended practice is to implement a two-step ownership transfer, as provided by the `OwnableTwoStepMixinImpl` component. This approach requires the new owner to explicitly accept the ownership before the transfer is finalized, reducing the risk of mistakenly assigning ownership to an incorrect address.

Recommendation(s): Consider replacing the current `OwnableMixinImpl` with `OwnableTwoStepMixinImpl` to introduce a safer, two-step ownership transfer process.

Status:Fixed

Update from client: 5cdb0d324566d83695a6ddc9c58ba232c6d15322

7 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 they provide a reference for developers who need to modify or maintain the contract in the future.

Remarks about Spiko documentation

The Spiko team provided comprehensive documentation that outlines the various user flows within the protocol.
Additionally, the Spiko team has effectively addressed concerns and questions raised by the Nethermind Security team during their regular calls.

8 Test Suite Evaluation

8.1 Compilation Output

```
> scarb build

Compiling snforge_scarb_plugin v0.1.0
 ↪  (git+https://github.com/foundry-rs/starknet-foundry?tag=v0.30.0#196f06b251926697c3d66800f2a93ae595e76496)
Finished `release` profile [optimized] target(s) in 0.30s
Compiling starknet_contracts v0.1.0 (/spiko/Scarb.toml)
Finished release target(s) in 56 seconds
```

8.2 Tests Output

```
> snforge test

Collected 23 test(s) from starknet_contracts package
Running 23 test(s) from tests/
[PASS] starknet_contracts_integrationtest::test_contract::admin_can_upgrade_permission_manager_contract (gas: ~310)
[PASS] starknet_contracts_integrationtest::test_contract::non_admin_can_not_upgrade_permission_manager_contract (gas:
 ↪  ~303)
[PASS] starknet_contracts_integrationtest::test_contract::non_owner_can_not_upgrade_redemption_contract (gas: ~172)
[PASS] starknet_contracts_integrationtest::test_contract::my_test (gas: ~1791)
[PASS] starknet_contracts_integrationtest::test_contract::minter_can_not_mint_token_if_paused (gas: ~1107)
[PASS] starknet_contracts_integrationtest::test_contract::can_set_name_symbol_and_decimals (gas: ~503)
[PASS] starknet_contracts_integrationtest::test_contract::non_executor_can_not_execute_redemption (gas: ~1934)
[PASS] starknet_contracts_integrationtest::test_contract::can_not_initiate_redemption_with_same_arguments_twice (gas:
 ↪  ~1812)
[PASS] starknet_contracts_integrationtest::test_contract::non_minter_can_not_mint_token (gas: ~930)
[PASS] starknet_contracts_integrationtest::test_contract::minter_not_whitelisted_by_whitelister_can_not_mint_token
 ↪  (gas: ~1010)
[PASS] starknet_contracts_integrationtest::test_contract::minter_whitelisted_by_whitelister_can_not_transfer_tokens
_to_non_whitelisted_user (gas: ~1319)
[PASS] starknet_contracts_integrationtest::test_contract::minter_whitelisted_by_whitelister_can_mint_token (gas: ~1260)
[PASS]
 ↪  starknet_contracts_integrationtest::test_contract::minters_whitelisted_by_whitelister_can_transfer_each_other_tokens
 ↪  (gas: ~1395)
[PASS] starknet_contracts_integrationtest::test_contract::minter_can_redeem_with_redemption_executed_by_executor (gas:
 ↪  ~1921)
[PASS] starknet_contracts_integrationtest::test_contract::minter_can_redeem_with_redemption_canceled_by_executor (gas:
 ↪  ~1971)
[PASS] starknet_contracts_integrationtest::test_contract::non_executor_can_not_cancel_redemption (gas: ~1858)
[PASS] starknet_contracts_integrationtest::test_contract::non_pauser_can_not_pause_token (gas: ~929)
[PASS] starknet_contracts_integrationtest::test_contract::non_owner_can_not_upgrade_token_contract (gas: ~496)
[PASS] starknet_contracts_integrationtest::test_contract::owner_can_upgrade_redemption_contract (gas: ~180)
[PASS] starknet_contracts_integrationtest::test_contract::owner_can_upgrade_token_contract (gas: ~505)
[PASS] starknet_contracts_integrationtest::test_contract::pauser_can_pause_and_unpause_token (gas: ~987)
[PASS] starknet_contracts_integrationtest::test_contract::redemption_can_not_be_canceled_twice (gas: ~1910)
[PASS] starknet_contracts_integrationtest::test_contract::redemption_can_not_be_executed_twice (gas: ~1857)
Running 0 test(s) from src/
Tests: 23 passed, 0 failed, 0 skipped, 0 ignored, 0 filtered out
```

Remarks about Spiko test suite

The Spiko team has developed a test suite that covers the protocol's basic flows. However, it currently does not include a variety of flows and execution scenarios. We recommend enhancing the test suite to cover all system specifications thoroughly.

9 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