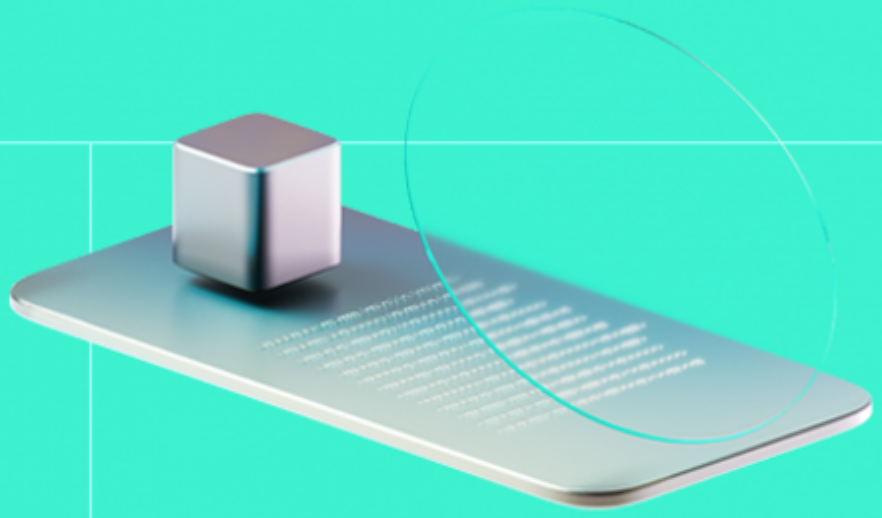
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Ledgity

Date: 18/03/2024

We express our gratitude to the Ledgity team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Ledgity Yield - is a stablecoin yield protocol backed by real-world assets (RWA). It aims to provide a stable and institutional-grade treasury management solution for stablecoins holders, and bridges the gap between DeFi and TradFi by offering real yield based on real assets.

Platform: EVM (Linea and Arbitrum)

Language: Solidity

Tags: Token Yield Farming

Timeline: 26/02/2024 - 18/03/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/LedgityLabs/LedgityYield
Commit	74ba73f9a81012e4a6040810b8f5413f7e2591e1

Audit Summary

10/10	10/10	100%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 10/10

The system users should acknowledge all the risks summed up in the risks section of the report

1	1	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	1

Vulnerability	Status
F-2024-1188 - Unrestricted Fees	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Ledgity
Audited By	Maksym Fedorenko, Roman Tiutiun
Approved By	Grzegorz Trawinski
Website	https://ledgity.finance/
Changelog	04/03/2024 - Preliminary Report, 18/03/2024 - Second Report

Table of Contents

System Overview

Ledgity Yield is a protocol that aims to provide stablecoins holders with a stable and scalable on-chain treasury management solution, backed by Real World Assets (RWA) with the following contracts:

- LToken.sol — is the main contract of the Ledgity Yield protocol as it powers every. It acts as a bridge between DeFi and TradFi worlds by allowing wallets to deposit stablecoins and Ledgity team to manage those stablecoins straightforwardly and securely.
- InvestUpgradeable.sol — is the derived contracts that are provided with utilities to manage an invested token, users' investment periods, rewards calculations, virtual balances, and auto-compounding.
- APRHistory.sol — is the managing the history of Annual Percentage Rates (APRs) on-chain. It uses a space-efficient storage pattern to minimize gas costs.
- ERC20BaseUpgradeable.sol — is an extension of BaseUpgradeable intended to be used as a base for ERC20 token contracts.
- BaseUpgradeable.sol — is an abstract contract that acts as a base for numerous contracts in this codebase, minimizing code repetition and enhancing readability and maintainability.
- SUD.sol — serves as an intermediary number format for calculations within this codebase. It ensures consistency and reduces precision losses. This library facilitates conversions between various number formats and the SUD format.
- GlobalPause.sol — is a contract that holds a global pause state for the entire Ledgity Yield ecosystem. This is very practical, because if a potentially dangerous situation for the protocol is detected, the whole protocol's ecosystem can be paused by a single on-chain transaction.
- GlobalOwnableUpgradeable.sol — is designed for upgradeable contracts to inherit a centralized ownership model from a specified GlobalOwner contract.
- GlobalBlacklist.sol — this contract maintains a global registry of blacklisted wallets. This is notably used for AML enforcement but also as a general way to recover funds originating from malicious operations (e.g., other protocol hacks).
- GlobalRestrictableUpgradeable.sol — is designed for security and upgradeability, with a focus on centralized blacklist management for a suite of contracts.
- GlobalPausableUpgradeable.sol — provides a centralized way to pause functionality across multiple contracts by referencing a shared GlobalPause contract.
- RecoverableUpgradeable.sol — is designed for asset recovery, specifically ERC20 tokens, in upgradeable contracts while ensuring ownership and proper initialization. It's part of a system that prioritizes security and future extensibility.
- LTokenSignaler.sol — Used to inform subgraph from the existence of a new L-Token contract. Once signaled, an L-Token will start being indexed.
- GlobalOwner — this contract holds the addresses of the Ledgity Yield protocol's owner (multisig), shared by all contracts of the ecosystem.
- DummyLDYStaking.sol — the contract acts as a placeholder for the real LDYStaking contract until this one is deployed.

Privileged roles

- The admin of the **`LToken.sol`** contract has several key responsibilities and capabilities:
  - Update the logic of the system.

    - Pause the transfers and withdrawals.
    - Change the user staking delegates.
    - Blacklist users.
    - Can claim unclaimed fees generated from successful withdrawals.
    - Can recover ERC20 tokens mistakenly sent to the contract (`recoverERC20`), except for the underlying token.
    - Can add (`listenToTransfers`) or remove (`unlistenToTransfers`) contracts that listen to L-Token transfers.
    - Can set parameters like withdrawal fees rate (`setFeesRate`), retention rate (`setRetentionRate`), and addresses for the LDYStaking contract (`setLDYStaking`), withdrawer wallet (`setWithdrawer`), and fund wallet (`setFund`).
- The Fund account processes big withdrawal requests and responsible for keeping `LToken` contract balance sufficient for the users withdrawals.
- The Withdrawer processes the queue of user transactions.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical requirements are provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage),

- The branch coverage ranges from 94%+ but never reaches 100%, due to false positives caused by the bugs with the `forge coverage` utility.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **1** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- Contracts might be upgraded after thr deployment, but these changes must be approached with caution as they can potentially introduce critical vulnerabilities.
- Owner might redirect rewards from and to any user.
- The system might be paused by the owner any time disallowing deposits and withdrawals.
- The LToken contract does not hold the significant part of the user deposited funds, **most of the underlying tokens are controlled by the Admin multisig contract** which is called Fund contract. Such behaviour matches requirements, so the admins (Fund account) is responsible for transfering the tokens to the main contract to cover withdrawal requests. The behaviour is explained by the client: `the Fund multisig will repatriate on the contract only the stablecoins that are needed to fulfill withdrawal requests, pay yield, plus a small reserve to enable instant withdrawal. This is pretty convenient as it consequently reduces the on-chain attack surface of the protocol`. The smart contract logic is not responsible for keeping smart contract solvent, it depends on the Admin behaviour.
- The fees which are applied during the instant withdrawals limited to 20%.

Findings

Vulnerability Details

F-2024-1188 - Unrestricted Fees - Low

Description:

The `LToken` contract currently does not enforce any limitations on the system withdrawal fees that can be applied, allowing for the possibility of any fees, including 100% or more. `feesRateUD7x3` is not restricted,

```
function setFeesRate(uint32 feesRateUD7x3_) public onlyOwner {
feesRateUD7x3 = feesRateUD7x3_;
}
```

This might lead to the denial of service if the value higher than 100% will be specified or to the users unexpected expenses if the fee is updated after the users deposit is made.

Assets:

- contracts/src/LToken.sol [https://github.com/LedgityLabs/LedgityYield]

Status: Fixed

Classification

Severity: Low

Recommendations

Recommendation: Add the validation to ensure that the fees are lower than the threshold, for example, 20%.

Remediation (revised commit: 3505bd): Proposed validation was added, the max withdrawal fee is 20%

Observation Details

F-2024-1160 - Initializer Is Not Disabled In Constructor - Info

Description: According to the OpenZeppelin documentation, upgradeable contracts should invoke the method `_disableInitializers()` in their `constructor()` to disable implementation contract, preventing them from being used or altered.

However, that functionality is not implemented in `LToken.sol` upgradeable contract.

Assets:

- contracts/src/LToken.sol [https://github.com/LedgityLabs/LedgityYield]

Status: Accepted

Recommendations

Recommendation: Follow OpenZeppelin's documentation regarding `_disableInitializers()` in `LToken.sol` upgradeable contract.

F-2024-1182 - Floating Pragma - Info

Description: The project uses floating pragmas `^0.8.18` in all contracts

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Status: Fixed

Recommendations

Recommendation: Consider locking the pragma version in all contracts.

Remediation (revised commit: 3505bd): Pragma version was locked to `0.8.18`.

F-2024-1183 - Missing Zero Address Validation - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "**zero address**". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The "**Missing zero address control**" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, consider a contract that includes a function to change its owner. This function is crucial, as it determines who has administrative access. However, if this function lacks proper validation checks, it might inadvertently permit the setting of the owner to the zero address. Consequently, the administrative functions will become unusable.

Function `setLDYStaking()`, `listenToTransfers()` and `unlistenToTransfers()` in `LToken.sol` are lack of missing zero address validation.

Status: Accepted

Recommendations

Recommendation:

Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

F-2024-1187 - Redundant Array For Frozen Request - Info

Description:

The contract has the `frozenRequests` array, which stores the requests from the blacklisted users. The array is permanently growing, and the data from the array are not utilised onchain.

This introduces the redundancy and redundant Gas expenses processing the users withdrawal requests.

Assets:

- contracts/src/LToken.sol [https://github.com/LedgityLabs/LedgityYield]

Status: Accepted

Recommendations

Recommendation:

If the `frozenRequests` array is needed for the tracking purpose, it is recommended to utilize the event emitting instead of occupying permanent storage.

F-2024-1190 - Missing Events Emitting For Critical Functions - Info

Description: The main `LToken` contract inherits the `GlobalBlacklist` contract responsible for blacklisting functionality. Adding or removing the users from the blacklist with the `blacklist`, `unBlacklist` functions introduces significant state changes which should be properly tracked offchain for the user experience convenience.

Assets:

- contracts/src/GlobalPause.sol [https://github.com/LedgityLabs/LedgityYield]

Status: Fixed

Recommendations

Recommendation: Consider emitting the corresponding events in the critical functions.

Remediation (revised commit: 6fd9d2): Events were added to `unBlacklist()` and `blacklist()` functions.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Primary Scope

Details

Repository	https://github.com/LedgityLabs/LedgityYield
Commit	74ba73f9a81012e4a6040810b8f5413f7e2591e1
Whitepaper	https://docs.ledgity.finance/architecture/core-contracts
Requirements	https://docs.ledgity.finance/architecture/core-contracts
Technical Requirements	SHA3: 5e57e2b7f45bbbce66b771941b3c61af992b98ea61352f885db2c1626fdd1a61 File: ledgity.gitbook.io.zip

Secondary

Scope Details

Repository	https://github.com/LedgityLabs/LedgityYield
Commit	6fd9d2123da96147f4919264b4b917e55385b839
Whitepaper	https://docs.ledgity.finance/architecture/core-contracts
Requirements	https://docs.ledgity.finance/architecture/core-contracts
Technical Requirements	SHA3: 5e57e2b7f45bbbce66b771941b3c61af992b98ea61352f885db2c1626fdd1a61 File: ledgity.gitbook.io.zip

Contracts in Scope

contracts/src/interfaces/ITransfersListener.sol
contracts/src/libs/APRHistory.sol
contracts/src/libs/SUD.sol
contracts/src/abstracts/InvestUpgradeable.sol

Contracts in Scope

contracts/src/abstracts/base/ERC20BaseUpgradeable.sol

contracts/src/abstracts/base/BaseUpgradeable.sol

contracts/src/abstracts/GlobalOwnableUpgradeable.sol

contracts/src/abstracts/RecoverableUpgradeable.sol

contracts/src/abstracts/GlobalRestrictableUpgradeable.sol

contracts/src/abstracts/GlobalPausableUpgradeable.sol

contracts/src/GlobalPause.sol

contracts/src/LTokenSignaler.sol

contracts/src/GlobalBlacklist.sol

contracts/src/GlobalOwner.sol

contracts/src/DummyLDYStaking.sol

contracts/src/LToken.sol