QuillAudits

AUDIT REPORT

May , 2025

For

Spheron

Table of Content

Table of Content 02

Executive Summary 03

Number of Issues per Severity 05

Checked Vulnerabilities 06

Techniques & Methods 08

Types of Severity 10

Types of Issues 11

Medium Severity Issues 12

1. withdrawEarnings can be initiated by anyone 12

Low Severity Issues 14

1. Use block.timestamp during signatures 14

Functional Tests 15

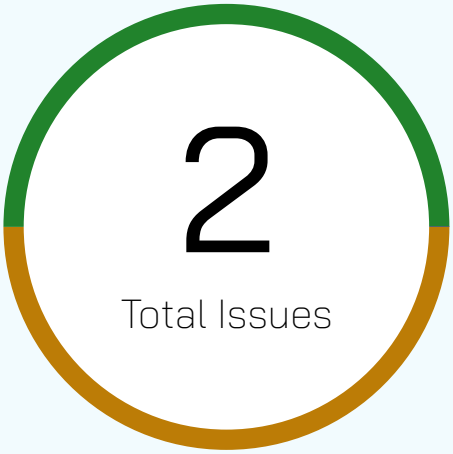
Closing Summary & Disclaimer 16

Executive Summary

Project name	spheronFdn
Project URL	https://www.spheron.network/
Overview	This is a lending pool for miners, enabling users to deposit mining tokens in exchange for pTokens. Implements borrowing and repayment mechanisms for agents, with interest accrual, liquidation processes, and distribution of rewards to token holders.
Audit Scope	The scope of this Audit was to analyze the spheronFdn Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/spheronFdn/protocol-contracts/tree/main
Contracts in Scope	contracts/compute/ComputeLease.sol src/payment/EscrowUser.sol src/payment/EscrowProtocol.sol
Branch	Main
Commit Hash	9aa8b7571d716c8235505ffcb2b4d8162e6a5412
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	27th March 2025 - 3rd April 2025

Updated Code Received	9th April 2025
Review 2	9th April 2025
Fixed In	https://github.com/spheronFdn/protocol-contracts/pull/57

Number of Issues per Severity



High	0 (0.00%)
Medium	1 (50.00%)
Low	1 (50.00%)
Informational	0 (0.00%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	**1**	**1**	0
Acknowledged	0	0	0	0
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✔ Access Management
- ✔ Arbitrary write to storage
- ✔ Centralization of control
- ✔ Ether theft
- ✔ Improper or missing events
- ✔ Logical issues and flaws
- ✔ Arithmetic Computations Correctness
- ✔ Race conditions/front running
- ✔ SWC Registry
- ✔ Re-entrancy
- ✔ Timestamp Dependence
- ✔ Gas Limit and Loops
- ✔ Exception Disorder
- ✔ Gasless Send
- ✔ Use of tx.origin
- ✔ Malicious libraries
- ✔ Compiler version not fixed
- ✔ Address hardcoded
- ✔ Divide before multiply
- ✔ Integer overflow/underflow
- ✔ ERC's conformance
- ✔ Dangerous strict equalities
- ✔ Tautology or contradiction
- ✔ Return values of low-level calls
- ✔ Missing Zero Address Validation
- ✔ Private modifier
- ✔ Revert/require functions
- ✔ Multiple Sends
- ✔ Using suicide
- ✔ Using delegatecall
- ✔ Upgradeable safety
- ✔ Using throw

- [x] Using inline assembly
- [x] Unsafe type inference
- [x] Style guide violation
- [x] Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools And Platforms Used For Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity statistical analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

Security vulnerabilities identified that must be resolved and are currently unresolved.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

Medium Severity Issues

withdrawEarnings can be initiated by anyone

Resolved

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/EscrowProtocol.sol#L149-L185

Function

withdrawEarnings

Description

The below function can be initiated by anyone by passing isFizz = true and _fizzId of anyone to initiate withdrawal without consent.The first if block will just transferEarnedTokens to the rewardWallet.
No loss of tokens since the transfer is done to the rewardWallet but withdrawals can be initiated without consent.

```solidity
   function withdrawEarnings(
        address _provider,
        uint256 _fizzId,
        address _token,
        uint256 _amount,
        bool isFizz
    ) external nonReentrant validAmount(_amount) {
        address rewardWallet;

        if (isFizz) {
            FizzTokenData storage fizz = fizzNodeEarnings[_fizzId][_token];
            if (_amount > fizz.balance)
                revert InsufficientBalance(fizz.balance, _amount);
            fizz.balance -= uint128(_amount);
            fizz.withdrawn += uint128(_amount);
            rewardWallet = fizzRegistry.getFizzNodeRewardWalletById(_fizzId);
            // send call to escrow to send fizz some token
            escrow.transferEarnedTokens(_token, rewardWallet, _amount);
            emit FizzMovement(_fizzId, _token, _amount);
        }
```

```diff
function withdrawEarnings(
        address _provider,
        uint256 _fizzId,
        address _token,
        uint256 _amount,
        bool isFizz
    ) external nonReentrant validAmount(_amount) {
        address rewardWallet;

        if (isFizz) {
+           if (
+           msg.sender != fizzRegistry.getFizzNodeRewardWalletById(_fizzId);
+           ) revert UnauthorizedAccess();

            FizzTokenData storage fizz = fizzNodeEarnings[_fizzId][_token];
            if (_amount > fizz.balance)
                revert InsufficientBalance(fizz.balance, _amount);

            fizz.balance -= uint128(_amount);
            fizz.withdrawn += uint128(_amount);
            rewardWallet = fizzRegistry.getFizzNodeRewardWalletById(_fizzId);
            // send call to escrow to send fizz some token
            escrow.transferEarnedTokens(_token, rewardWallet, _amount);
            emit FizzMovement(_fizzId, _token, _amount);
        }
```

Low Severity Issues

Use block.timestamp during signatures

Resolved

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/EscrowUser.sol#L289-L303

Function

withdrawWithSignature

Description

:There should be a deadline set during signature verification otherwise the signature can be used for an indefinite period of time.

```solidity
// @audit no block.timestamp used in signature
    function withdrawWithSignature(
        address _token,
        uint256 _amount,
        address _operator,
        address claimedSigner,
        uint256 nonce,
        bytes memory signature
    ) external nonReentrant validAmount(_amount) {
        if (nonce != nonces[claimedSigner]) revert InvalidNonce();
        nonces[claimedSigner]++;

        // Create the struct hash based on the withdraw details
        bytes32 structHash = keccak256(
            abi.encode(WITHDRAW_TYPEHASH, _token, _amount, _operator, nonce)
        );
```

Functional Tests

Some of the tests performed are mentioned below:

- ✔ withdrawEarnings can be initiated by anyone

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Spheron changes Review. We performed our audit according to the procedure described above.

Issues of medium and low severity were found. Spheron team resolved them all

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

May , 2025

For

Spheron

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com