QuillAudits

AUDIT REPORT

May , 2025

For

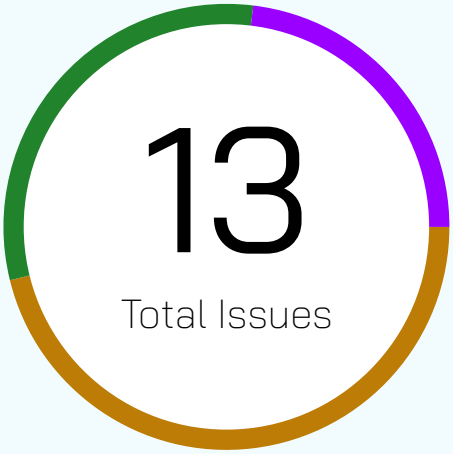
Spheron

Table of Content

Executive Summary

Project name	Spheron
Project URL	https://www.spheron.network/
Overview	Spheron Network is a decentralized compute platform designed to connect GPU providers with developers and businesses in need of computational resources.
Audit Scope	The Scope of the Audit was to analyse the Code security,Quality and correctness of Spheron Contracts.
Contracts in Scope	contracts/compute/ComputeLease.sol contracts/payment/Escrow.sol
Commit Hash	c6dbac74bfa15cb289cc5f3139ad4d1bae8c14df
Language	Solidity
Blockchain	EVM
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	11th october 2024 - 21st october 2024
Updated Code Received	NA
Review 2	NA
Fixed In	NA

Number of Issues per Severity



High	0 (0.00%)
Medium	6 (46.15%)
Low	4 (30.77%)
Informational	3 (23.08%)

Severity

Issues	High	Medium	Low	Informational
Open	0	0	0	0
Resolved	0	0	0	0
Acknowledged	0	**6**	**4**	**3**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✔ 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
- ✔ Using inline assembly
- ✔ Style guide violation
- ✔ Unsafe type inference
- ✔ 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

`createLease() & closeAndRenewLease()` would fail whenever it get's called

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/compute/Compute-Lease.sol#L81-L89

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/compute/Compute-Lease.sol#L213

Function

createLease() and closeAndRenewLease()

Description

Taking Example of createLease()(same applies for closeAndRenewLease)

- Whenever `createLease()` gets called the function would throw error.

- `createLease()` gets called from `Bid.sol::matchOrder()`.

- In `createLease()` there is a `onlyBidContract()` modifier which passes as msg.sender is Bid contract. Inside `createLease()`,`escrow.lockToken()` gets called : `escrow.lockTokens(order.creator, order.token, _toPrecision(leaseCostPlusFee, token.decimals));`

- `lockTokens()` contains a `onlyRole(DEFAULT_ADMIN_ROLE)` which means only admin can call this function but in our case `msg.sender` would be computeLease contract address so `lockTokens()` function will fail due to the `onlyRole(DEFAULT_ADMIN_ROLE)` modifier.

Impact

`createLease()` would never get executed and would fail every time

```
- `function matchOrder(uint64 _orderId, address _providerAddress, uint256 _acceptedPrice) public onlyAdmin returns(uint256)`

`function createLease(uint64 _orderId, uint256 _acceptedPrice, address _providerAdress, uint256 _fizzId) external onlyBidContract returns(uint256)`

`function lockTokens(address _user, address _token, uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE)`
```

Recommendation
Should check that who originated the transaction instead of checking msg.sender in modifier

Spheron team's Comment
This is not a valid issue and should not be classified as High severity. The initial claim was that the createLease() and closeAndRenewLease() functions would consistently fail, but this is incorrect. The contract deployed on the testnet operates as expected, and the functions execute without failure. The ComputeLease contract has the necessary admin permissions to call functions like lockTokens(), unlockTokens(), transferLocked(), and transferLockedFizz(), ensuring smooth transaction execution.

Provider would not able to withdraw earnings if providerRewardWallet is contract addresss

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L144

Function

withdrawProviderEarnings() and addProvider()

Description

Provider would not be able to claim earnings if providerRewardWallet is set as contract address.Let's suppose a provider has added himself into registry and has set providerRewardWallet to contract address where he wants the fund.

Now when he wants to claim his earnings via Escrow.withdrawProviderEarnings() he would not be able to claim due to the below check
`require(msg.sender == providerRewardWallet, "Not Provider Reward Wallet");`
So due to above check the provider would not be able to claim as contract address doesn't have `withdrawProviderEarnings` function selector and whenever the provider calls `withdrawProviderEarnings()` from his address it would revert due to above condition.
`"Not Provider Reward Wallet"`

```
function addProvider(
        ProviderParams memory params
    ) public {
        // Verify each payment method corresponds to a registered token
        for (uint256 i = 0; i < params.paymentsAccepted.length; i++) {
            if (!tokenRegistry.isAcceptedToken(params.paymentsAccepted[i])) {
                revert UnregisteredToken();
            }
        }
        if (isRegisteredProvider(params.walletAddress)) {
            revert ProviderAlreadyRegistered();
        }
        uint256 currentProviderId = nextProviderId;
        nextProviderId++;
        providers[currentProviderId] = Provider({
            providerId: currentProviderId,
            spec: params.spec,
            hostUri: params.hostUri,
            certificate: params.certificate,
            walletAddress: params.walletAddress,
            paymentsAccepted: params.paymentsAccepted,
            status: ProviderStatus.Registered,
            tier: TrustTier.Seven,
            joinTimestamp: block.timestamp,
            rewardWallet: params.rewardWallet
        });
        addressToProviderId[params.walletAddress] = currentProviderId;
        rewardAddressToProviderId[params.rewardWallet] = currentProviderId;

        emit ProviderAdded(currentProviderId, params.walletAddress, params.spec, params.hostUri,
        params.certificate, params.paymentsAccepted, ProviderStatus.Registered, TrustTier.Seven,
        block.timestamp, params.rewardWallet);
    }
```

Impact

Provider would not be able claim his earnings and earnings would stay in contract itself

Recommendation

One of the solution is to check the `providerRewardWallet` is not set as contract address.Another solution is allow only provider to withdraw his earnings and make an extra argument in `withdrawProviderEarnings()` at which address he to wants to receive his earnings.

Spheron Team's Comment

a provider can always withdraw their earnings to a contract address, provided the contract is properly set up with an interface to the Escrow contract and can call escrow.withdrawProviderEarnings(). In cases where a provider mistakenly sets the reward wallet to a contract that does not support this interface, they can easily update it using the updateProviderRewardAddress() function, allowing them to change the address to either a properly configured contract or an EOA.

Given these safeguards, this should not be classified as an issue.

No pause state checked during `withdrawProviderEarnings()`,`withdrawFizzNodeEarnings()` and `withdrawProtocolFees`

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L141C14-L163

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L171

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L200

Function

`withdrawProviderEarnings()`,`withdrawFizzNodeEarnings()` and `withdrawProtocolFees`

Description

There is no check while calling `withdrawProviderEarnings()` , `withdrawFizzNodeEarnings()` and `withdrawProtocolFees` that the withdrawls are in pause state or not.If `pauseWithdrawal` has been set to true there should be no withdrawals from the protocol.But by calling `withdrawProviderEarnings()` , `withdrawFizzNodeEarnings()` and `withdrawProtocolFees` they can withdraw funds even though the `pauseWithdrawal` has been set to true.It breaks the core functionality of the protocol.

Impact

Even when withdrawal is paused people can withdraw using the above mentioned functions which breaks the core functionality

Recommendation

The above two functions should have a check like :
`require(!pauseWithdrawal, "Withdrawal is paused");`
- The above check is important to halt every type of withdrawal when the withdrawals are paused.

Spheron Team's Comment

Good point raised. The design intentionally restricts user deposits and withdrawals during a paused state, while allowing providers and fizz nodes to withdraw their earnings without interruption. This ensures that providers and fizz nodes are not blocked from accessing their rewards, even when user operations are paused.

Malicious User would pay less tokens to provider under certain conditions

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/compute/Compute-Lease.sol#L248-L249

Function

_processLeasePay()

Description

Malicious User would pay less tokens even if he has used the service provider by provider more than the specified time in the order.

Lets understand using an example:

1. There is a matched order between user and provider and the conditions are `order.numOfBlocks = 100` and `acceptedPrice = 1e18`
2. Let's say after passing of 200 blocks the provider calls `closeLease()` to close the lease.
3. As user has used the service for 200 blocks he should pay tokens for 200 blocks.
4. But user would only pay the tokens for 100 blocks of service.
6. So it would be loss to provider as he will not receive tokens for remaining 100 blocks.
7. According to our example loss of 100e18 to provider.

Impact

Provider would not get extra tokens for extra amount of service used by user.

Recommendation

Provider should get tokens according to total time the service used by user and not according to numOfBlocks in order.

Spheron Team's Comment

While your point makes logical sense, the responsibility for closing a lease on time lies with the provider. If a provider fails to close the lease when due, it is to their detriment, as they are accountable for managing their own operations. Additionally, the system includes a checker that notifies providers when a lease is due for closure, ensuring they are reminded to take action.

Malicious Admin can steal away provider earnings

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/compute/Compute-Lease.sol#L247

Function

_processLeasePay()

Description

Malicious Admin can steal away provider earnings Whenever provider or user calls `closeLease()`, malicious admin can frontrun this transaction and increase the `protocolPerc` to a very high value such that the very major part of tokens as fees.

Now as protocolPerc has been set to almost 99% using`setProviderFeePercentage()` So provider would get very less amount of tokens as earnings and other major amount (99%) goes as fees to protocol.

Here providerAmount would be only 1% of total amount and protocolFee would 99% of total amount.

```solidity
(uint32 protocolPerc, uint32 userFeePerc) = escrow.getProtocolFee();
        uint256 totalAmount = _toPrecision((lowerValue * lease.acceptedPrice * (100 + userFeePerc)) / 100,
        token.decimals);
        uint256 providerAmount = _toPrecision((lowerValue * lease.acceptedPrice * (100 - protocolPerc)) /
        100, token.decimals);
        uint256 protocolFee = totalAmount - providerAmount;
```

Impact

Malicious Admin would steal provider's earnings.

Recommendation

There should be a max cap above which admin can't set the fees above that threshold.

Speheron Team's Comment

This is not considered an issue, as access to adjust protocol fees is strictly controlled. The use of a multi-signature wallet, requiring approval from both the CEO and CTO, ensures that any changes are securely governed, preventing any malicious interference.

Fee on Transfer Tokens doesn't work with `Escrow.sol`

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L92

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L104

Function

deposit()

Description

Some tokens have a fee on transfer, for example USDT. Usually such fee is not enabled but could be re-enabled at any time.
With this fee enabled the deposit() and withdraw() function would receive slightly less tokens than the amounts requested in the function parameter. This will create an accounting problem, as the whole `amount` variable gets added in unlock Tokens mapping but actually it will receive slightly less tokens than `amount`.
So in mapping it should add the actual received amount.

As main functionality is dependent on Lock and Unlock Tokens this should be handled correctly

```solidity
erc20.safeTransferFrom(msg.sender, address(this), _amount);
userBalance[msg.sender][_token].unlockedBalance += _amount;
```

Impact

Wrong Accounting of unLock and lockTokens in mapping
`userBalance[msg.sender][_token].unlockedBalance += _amount;`

Recommendation

Check the balanceOf() tokens before and after a safeTransfer() or safeTransferFrom().Use the difference as the amount of tokens sent/received.

Spheron Team's Comment

This is a valid observation. We acknowledge the potential for an accounting discrepancy when handling tokens with transfer fees, such as USDT, if the fee is enabled. To address this, we will ensure the contract properly accounts for the actual tokens received by checking the balance before and after the transfer.

Low Severity Issues

Unnecessary check of allowance in `deposit()`

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L101

Function

deposit()

Description

There is an unnecessary check of allowance in deposit() which is not at all required.

safeTransferFrom() have a built-in check that the user has provided an allowance or not.
So
`require(erc20.allowance(msg.sender, address(this)) >= _amount, "Insufficient allowance");`
This check is unnecessary and of no use. If there was no allowance check and the user did not approve then at that time also it would revert as the user hasn't provided approval. Because of this reason the check is unnecessary.

```solidity
require(erc20.allowance(msg.sender, address(this)) >= _amount, "Insufficient allowance");
erc20.safeTransferFrom(msg.sender, address(this), _amount);
```

Recommendation

Remove this check
`require(erc20.allowance(msg.sender, address(this)) >= _amount, "Insufficient allowance");`

Wrong data passed during event emission

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L232

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L245

Function

lockTokens() and unlockTokens()

Description

Wrong data passed during event emission in `lockTokens()` and `unlockTokens()`

As we can see during event emission in first parameter it should be `_user` address instead of `msg.sender`

Due to this wrong data gets passed during event emission

```
```solidity
  function lockTokens(address _user, address _token, uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE)
  {
        require(userBalance[_user][_token].unlockedBalance >= amount, "Insufficient unlocked balance");
        userBalance[_user][_token].unlockedBalance -= amount;
        userBalance[_user][_token].lockedBalance += amount;
        // @audit Low this should not be msg.sender but instead _user address
        emit UserTokenLocked(msg.sender, _token, amount);
    }
  ```
```solidity
  function unlockTokens(address _user, address _token, uint256 amount) external
  onlyRole(DEFAULT_ADMIN_ROLE) {
        require(userBalance[_user][_token].lockedBalance >= amount, "Insufficient locked balance");
        userBalance[_user][_token].lockedBalance -= amount;
        userBalance[_user][_token].unlockedBalance += amount;
        emit UserTokenUnlocked(msg.sender, _token, amount);
    }
  ```
```

Recommendation

Remove the above event emission and add the below event emission
`emit UserTokenLocked(_user, _token, amount);`

zero address check for providerRegistry not done

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L304-L307

Function

updateProviderRegistryAdress and updateFizzRegistryAdress

Description

The above function doesn't check if the new providerRegistry is a 0 address or not, just like there is a check in updateTokenRegistryContract function for 0 address , there should be a check in the updateProviderRegistryAdress function too
Same 0 address check should be included for updateFizzRegistryAdress too

```solidity
function updateProviderRegistryAdress(address _providerRegistry) onlyRole(DEFAULT_ADMIN_ROLE) external {
        providerRegistry = IProviderRegistry(_providerRegistry);
        emit ProviderRegistryUpdated(_providerRegistry);
    }
```

Recommendation

Add zero address check for both functions

Use of redundant functions

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L304-L316

Function

updateProviderRegistryAdress and updateProviderAdress

Description

The below two functions are used to do the same thing i.e update the provider registry address, one of these functions are redundant and can be removed

```solidity
/**
     * @notice Update the provider registry contract address.
     * @param _providerRegistry New address of the provider registry contract
    */
    function updateProviderRegistryAdress(address _providerRegistry) onlyRole(DEFAULT_ADMIN_ROLE) external
    {
        providerRegistry = IProviderRegistry(_providerRegistry);
        emit ProviderRegistryUpdated(_providerRegistry);
    }

    /**
     * @notice Update the provider registry contract address.
     * @param _providerRegistry New address of the provider registry contract
    */
    function updateProviderAdress(address _providerRegistry) onlyRole(DEFAULT_ADMIN_ROLE) external {
        providerRegistry = IProviderRegistry(_providerRegistry);
        emit ProviderRegistryUpdated(_providerRegistry);
    }
```

Recommendation

Remove one of the two functions

Informational Severity Issues

Wrong comment in function changeWithdrawal-Status()

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L336

Function

changeWithdrawalStatus

Description

changeWithdrawalStatus() function is used to change the withdrawal status but the comment says `@notice change status for user deposit. On or off` which should be updated to `@notice change status for user withdrawal. On or off`

Recommendation

update the comment to `@notice change status for user withdrawal. On or off`

Avoid Hardcoded address in contract

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/compute/Compute-Lease.sol#L39

Description

- Hardcoding any address in contract is bad practice.
- If we want to change the address in future we can't change it

`IArbSys public constant ARB_SYS = IArbSys(0x0000000000000000000000000000000000000064);`

Add a variable in constructor to set it and add a setter function whenever you want to change the address

Usage of Pausable library for pausing and unpausing during `deposit()` & `withdraw()`

Acknowledged

Path

https://github.com/spheronFdn/protocol-contracts/blob/main/contracts/payment/Escrow.sol#L41-L42

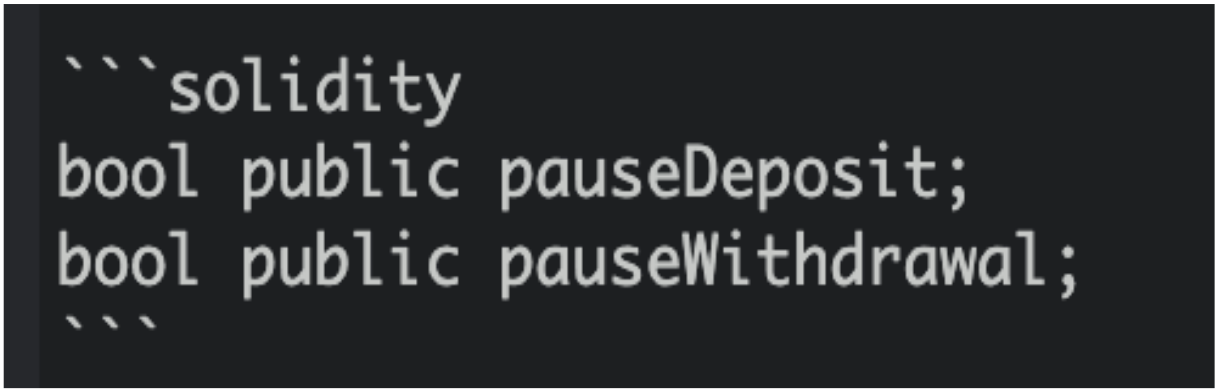
Function

bool public pauseDeposit;
bool public pauseWithdrawal;

Description

There are 2 variables which check if it is in a pause state or not.

Instead of using 2 variables only 1 variable can be used overall to check if the protocol is in pause state or not.
It is preferred to use the Pausable library.



Recommendation

Use Pausable library of Openzeppelin

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Pausable.sol

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. We performed our audit according to the procedure described above.

Some issues of low,medium and informational severity were found. Some suggestions, gas optimizations and best practices are also provided in order to improve the code quality and security posture.In the end,Spheron Team has Acknowledged all issues.

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