Cyberscope

Audit Report

Sphere

September 2023

Repository https://github.com/solstarter-org/sphere-staking-ethereum/blob/master/contracts/Staking.sol

Commit 6808b08c619d0ec3143d16c09ced42e09b222aee

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/solstarter-org/sphere-staking-ethereum/blob/master/contracts/Staking.sol
Commit	6808b08c619d0ec3143d16c09ced42e09b222aee
Testing Deploy	https://testnet.bscscan.com/address/0x6b92cdb3d1925d9cc6267ac0eda04342f1ba2941

Audit Updates

Initial Audit	15 Sep 2023 https://github.com/cyberscope-io/audits/blob/main/1-sphere/v1/audit.pdf
Corrected Phase 2	25 Sep 2023

Source Files

Filename	SHA256
contracts/Staking.sol	bd8e02526fd3704d31d562dc25ddab4cc24fd5af2e9010cb393e485953e55586
@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol	5b478023a1200e1364308ca06cdefec7cb7ab990a1cb904cbbdbaa7ba85076be
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	db92fc1b515decad3a783b1422190877d2d70b907c6e36fb0998d9465aee42db
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	04338003a3be8f5f38595048b591d80fdc147bf95cc7c6285e1e1a5f1afa2b47

@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	a2c4e5c274a586f145d278293ae33198cd8f412ab7e6d26f2394c8949b32b24b
@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol	e0ac7115916f0dce0a8e80769694736f3e674bdc5b2e5853964c82004b1e1cc5
@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol	40dd5b14a370eea51ba94eb1b66a89638c6c54d86cc9f406599075c273e5e4c6
@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol	a94576fd98585c07b2a9725f7c89c910a3a1909a03f49ec2df465327c6a0ffc3
@openzeppelin/contracts-upgradeable/interfaces/IERC1967Upgradeable.sol	167828e6f725b1d47d82bc912fd0f1c6ed0fb67a4e5e06a4d62e72b4a53e95cf
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	1fbf2a131b895514f0027866cc0deff151ea16424b4aed2b8c573d2275cfa9e8
@openzeppelin/contracts/utils/Address.sol	8b85a2463eda119c2f42c34fa3d942b61aee65df381f48ed436fe8edb3a7d602
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	c8309ed2c1c7edf52a23833798c94507702248debfc4ed1f645e571e3c230f8b
@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol	b7383c48331f3cc9901fc05e5d5830fcd533699a77f3ee1e756a98681bfbb2ee

Overview

The `Staking` contract is a staking that allows users to stake tokens in various pools to earn `stakingToken` as a reward. The contract is upgradeable, meaning that its logic can be updated by the contract owner. It also uses OpenZeppelin's libraries for standard functionalities like ownership, reentrancy guard, and safe math operations. The contract has an early pool with specific standards and the contract allows the owner to add more pools with different configurations. The contract also has admin functionalities to update the pool information and withdraw the `stakingToken`.

Stake Functionality

The `stake` function enables users to deposit `stakingToken` into a designated pool, identified by its `pid` number. After validating the pool index and ensuring the staking amount is within the pool's minimum and maximum limits, the function proceeds to handle additional conditions. In the case of the early pool (where pid == 0), staking is permitted only within a specific timeframe since the contract's initialization. The function also checks that the user has not already staked an amount in the specific `pid` pool. The function updates both the pool's and the user's information, including their staked balances and the last time of staking. The staking tokens are then transferred from the user's address to the contract, and the pool's balance and reward allocations are decreased accordingly. Rewards are calculated based on the staked amount and the pool's APY percentage. Finally, an event is emitted to log the staking action.

Unstake Functionality

The `unstake` function enables users to withdraw their staked tokens and collect any applicable rewards. It first verifies that the user has tokens staked and that the pool's freeze time has elapsed since the last staking event. The function then checks whether the `poolLockupSeconds` duration has passed. If not, the function proceeds to unstake only the original staked amount, excluding any additional rewards. However, if the `poolLockupSeconds` duration has passed, rewards are calculated based on the pool's APY and added to the staked amount. Subsequently, the function updates both the pool's and the user's information, including staked balances. The staked tokens, along with any

earned rewards if applicable, are then transferred back to the user's address. Finally, an event is emitted to log the unstaking action.

Roles

Owner

The owner has the authority to initialize the Staking contract. The owner is responsible for setting up the staking token, early pool information, pool lifetime, and pool freeze time.

The owner can interact with the following functions:

- function Staking_init(address token, PoolInfo calldata earlyPoolData, uint256 lifeTime, uint256 freezeTime)
- function adminWithdraw(uint256 amount)
- function setPoolFreezeTime(uint256 freezeTime)
- function addPoolInfo(PoolInfo calldata newPoolData)
- function setPoolInfo(uint256 pid, PoolInfo calldata newPoolData)

User

The user can interact with the following functions:

- function stake(uint8 pid, uint256 amount)
- function unstake(uint8 pid)
- function readPoolInfo(uint8 pid)
- function getUserInfo(uint8 pid, address user)
- function getUserTotalBalance(address user)
- function getPoolLength()

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	0	7	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CR	Centralization Risk	Acknowledged
	OCTD	Transfers Contract's Tokens	Acknowledged
	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
	MVN	Misleading Variables Naming	Acknowledged
	RCI	Reward Calculation Inconsistency	Acknowledged
	RSK	Redundant Storage Keyword	Acknowledged
	L04	Conformance to Solidity Naming Conventions	Acknowledged

CR - Centralization Risk

Criticality	Minor / Informative
Location	Staking.sol#L75,209,220
Status	Acknowledged

Description

The contract poses a centralization risk. Specifically, the `Staking_init` function allows the contract owner to initialize critical parameters like the `stakingToken`, the initial PoolInfo for early pool data, as well as important time-related variables such as `lifeTime` and `freezeTime`. This poses a significant risk to the overall security and decentralized nature of the contract.

In addition to setting the initial parameters, the contract owner also has the authority to transfer the `stakingToken` amount to the contract which will be used as rewards.

Furthermore, the contract owner can set any variable within the `PoolInfo` struct, enabling the owner to add new pools or update existing ones. The contract owner can modify key attributes such as `poolLockupSeconds`, `stakeMinAmount`, `stakeMaxAmount`, `poolRewardRate`, `poolBalanceAllocation`, and `poolRewardAllocation`. This level of control held by a single entity undermines the trustless and decentralized attributes generally expected of blockchain-based systems.

Additionally, the contract is designed to allow the addition and modification of staking pools through addPoolInfo and setPoolInfo functions. These functions take a PoolInfo struct as an argument and set the values for poolBalanceAllocation and poolRewardAllocation without any constraints or validation checks. While this provides maximum flexibility, it poses a considerable risk as the contract owner can set these variables to unreasonable values. Such an action could inadvertently disable the staking functionality for the users, leading to a lack of trust and possible financial consequences.

```
function Staking_init(
        address token,
        PoolInfo calldata earlyPoolData,
        uint256 lifeTime,
        uint256 freezeTime
    ) public initializer {
        stakingToken = IERC20(token);
        pool0CreatedTime = block.timestamp;
        poolInfo.push(earlyPoolData);
        pool0LifeTime = lifeTime;
        poolFreezeTime = freezeTime;
        __Ownable_init();
        __UUPSUpgradeable_init();
    }

    function addPoolInfo(PoolInfo calldata newPoolData) public onlyOwner
{
        require(newPoolData.stakeMinAmount < newPoolData.stakeMaxAmount,
"Wrong amount range");
        require(poolFreezeTime < newPoolData.poolLockupSeconds, "Wrong
LockupSeconds");
        poolInfo.push(newPoolData);
    }

    function setPoolInfo(
        uint8 pid,
        PoolInfo calldata newPoolData
    ) public validPID(pid) onlyOwner {
        require(newPoolData.stakeMinAmount < newPoolData.stakeMaxAmount,
"Wrong amount range");
        require(poolFreezeTime < newPoolData.poolLockupSeconds, "Wrong
LockupSeconds");
        poolInfo[pid] = newPoolData;
   }
```

```
   struct PoolInfo {
        ...
        uint256 poolBalanceAllocation;  // pool available balance
        uint256 poolRewardAllocation;   // pool reward allocation
        uint256 poolStakedBalance;      // pool stake balance
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

Additionally, the team is advised to properly check the variables according to the required specifications. It is recommended to incorporate additional checks before setting or updating the `poolBalanceAllocation` and `poolRewardAllocation` variables. One suggestion is to enforce a lower limit on the `poolBalanceAllocation` variable that cannot be less than the current `poolStakedBalance`. Similarly, the `poolRewardAllocation` should not be set lower than the reward allocation already used, preventing potential imbalances and ensuring the staking and unstaking mechanisms operate as expected. This will add an extra layer of security and robustness to the contract.

Team Update

The team has acknowledged that this is not a security issue.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	Staking.sol#L200
Status	Acknowledged

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `adminWithdraw` function.

```
    function adminWithdraw(uint256 amount) public virtual onlyOwner {
        stakingToken.safeTransfer(msg.sender, amount);
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

Team Update

The team has acknowledged that this is not a security issue.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	Staking.sol#L117
Status	Acknowledged

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
        stakingToken.safeTransferFrom(
                address(msg.sender),
                address(this),
                amount
            );

            poolInfo[pid].poolStakedBalance =
poolInfo[pid].poolStakedBalance.add(
                amount
            );
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

Team Update

The team has acknowledged that this is not a security issue.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	Staking.sol#L193
Status	Acknowledged

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

Specifically, the contract is using the `poolFreezeTime` variable, which is used to enforce a minimum time before unstaking, can be updated by calling the `setPoolFreezeTime` function. This contradicts the documentation, which suggests that `poolFreezeTime` is a fixed number. As a result, `poolFreezeTime` does not represent a fixed number and can change, leading to potential inconsistencies and unexpected behavior.

```
require(block.timestamp - user.lastStakeTime >= poolFreezeTime, "Pool
Freeze Time");
...
function setPoolFreezeTime(uint256 freezeTime) public onlyOwner {
    ...
    poolFreezeTime = freezeTime;
}
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code. It is recommended to use a fixed number for the `poolFreezeTime` variable as suggested in the documentation, rather than allowing it to be updated dynamically.

Team Update

The team has acknowledged that this is not a security issue.

RCI - Reward Calculation Inconsistency

Criticality	Minor / Informative
Location	Staking.sol#L99,138
Status	Acknowledged

Description

The contract is designed to handle staking and unstaking of tokens with associated rewards. Within the `_stake` function, it calculates the `rewardAmount` based on the `poolRewardRate` at the time of staking and updates `poolRewardAllocation` accordingly. Similarly, in the `_unstake` function, the `rewardAmount` is again calculated based on the `poolRewardRate`, but this time at the moment of unstaking. However, the contract does not account for the possibility that the `poolRewardRate` may have changed between the staking and unstaking events.

This implementation can lead to inconsistencies and incorrect calculations in the `poolRewardAllocation`. For example, if the `poolRewardRate` has increased since the user staked their tokens, the unstake function would calculate a higher `rewardAmount` than was originally allocated during staking, which could result in an imbalance in the `poolRewardAllocation`.

```
function _stake(uint8 pid, uint256 amount) internal {
        ...
        uint256 rewardAmount = amount * (poolInfo[pid].poolRewardRate) /
10000;
        ...
        poolInfo[pid].poolRewardAllocation -= rewardAmount;
        ...
    }

    function _unstake(uint8 pid) internal {
        ...
        uint256 rewardAmount = user.amount *
poolInfo[pid].poolRewardRate / 10000;

        if (block.timestamp - user.lastStakeTime <
poolInfo[pid].poolLockupSeconds) {
        ...
            poolInfo[pid].poolRewardAllocation += rewardAmount;
        } else {
            // add reward when unstake after lockup period
            unstakeAmount += rewardAmount;
        }
        ...

        stakingToken.safeTransfer(address(msg.sender), unstakeAmount);
        ...
    }
```

Recommendation

It is recommended to add a separate state variable or mapping to keep track of the `rewardAmount` allocated to each user at the time of staking. This ensures that the same `rewardAmount` is used for both staking and unstaking actions, regardless of any changes in the `poolAPY`.

Additionally, consider updating the logic in the `_unstake` function to refer to the `rewardAmount` captured at the time of staking, rather than recalculating it based on the potentially changed `poolAPY`.

By doing these, the code can prevent the inconsistencies and potential imbalances in the `poolRewardAllocation`, making the contract's behavior more predictable and robust.

Team Update

The team has acknowledged that this is not a security issue.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/Staking.sol#L261
Status	Acknowledged

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
UserInfo storage userinfo
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/Staking.sol#L75
Status	Acknowledged

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function Staking_init(
        address token,
        PoolInfo calldata earlyPoolData,
        uint256 lifeTime,
        uint256 freezeTime
    ) public initializer {
        stakingToken = IERC20(token);
        pool0CreatedTime = block.timestamp;
        poolInfo.push(earlyPoolData);
        pool0LifeTime = lifeTime;
        poolFreezeTime = freezeTime;
        __Ownable_init();
        __UUPSUpgradeable_init();
    }
```

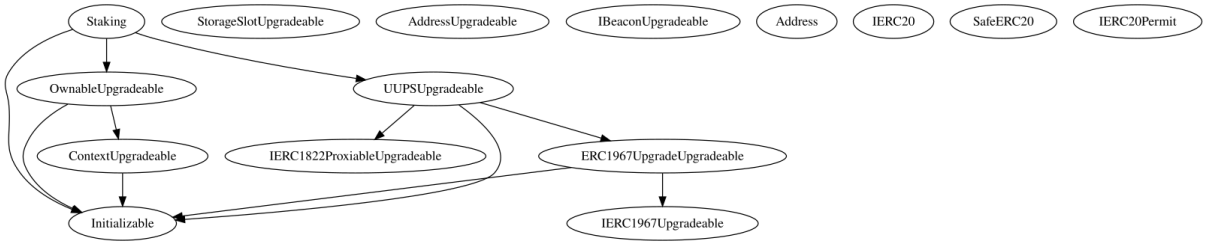
Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

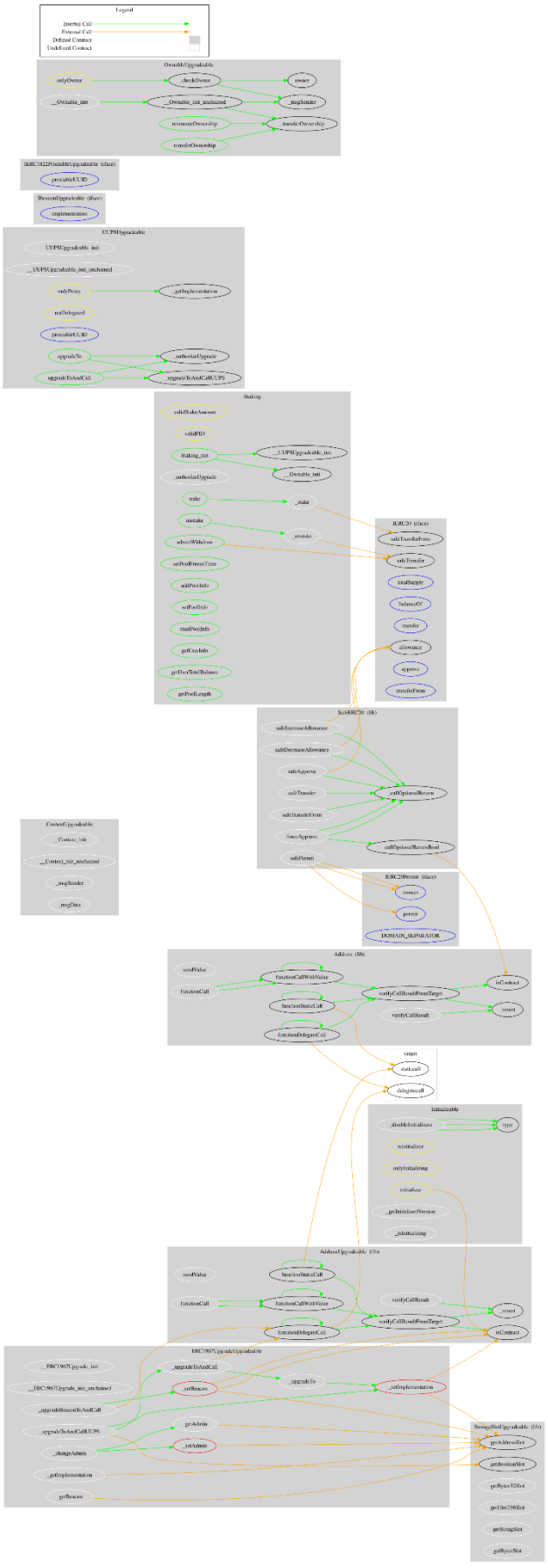
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Staking	Implementation	Initializable, UUPSUpgradeable, OwnableUpgradeable		
	Staking_init	Public	✓	initializer
	_authorizeUpgrade	Internal	✓	onlyOwner
	_stake	Internal	✓	
	_unstake	Internal	✓	
	stake	Public	✓	validPID validStakeAmount
	unstake	Public	✓	validPID
	adminWithdraw	Public	✓	onlyOwner
	setPoolFreezeTime	Public	✓	onlyOwner
	addPoolInfo	Public	✓	onlyOwner
	setPoolInfo	Public	✓	validPID onlyOwner
	readPoolInfo	Public		validPID
	getUserInfo	Public		-
	getUserTotalBalance	Public		-
	getPoolLength	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
OwnableUpgradeable
onlyOwner
checkOwner
owner
__Ownable_init
__Ownable_init_unchained
_msgSender
renounceOwnership
transferOwnership
IERC1822ProxiableUpgradeable (iface)
proxiableUUID
IBeaconUpgradeable (iface)
implementation
UUPSUpgradeable
UUPSUpgradeable_init
__UUPSUpgradeable_init_unchained
onlyProxy
_getImplementation
notDelegated
proxiableUUID
upgradeTo
upgradeToAndCall
_authorizeUpgrade
_upgradeToAndCallUUPS
Staking
validStakeAmount
validPID
Staking_init
__UUPSUpgradeable_init
__Ownable_init
_authorizeUpgrade
stake
_stake
unstake
_unstake
adminWithdraw
setPoolFreezeTime
addPoolInfo
setPoolInfo
modPoolInfo
getUserInfo
getUserTotalBalance
getPoolLength
IERC20 (iface)
safeTransferFrom
safeTransfer
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
SafeERC20 (lib)
safeIncreaseAllowance
safeDecreaseAllowance
safeApprove
safeTransfer
safeTransferFrom
forceApprove
callOptionalReturn
callOptionalReturnBool
safePermit
IERC20Permit (iface)
nonces
permit
DOMAIN_SEPARATOR
ContextUpgradeable
__Context_init
__Context_init_unchained
_msgSender
_msgData
Address (lib)
sendValue
functionCall
functionCallWithValue
functionStaticCall
functionDelegateCall
verifyCallResultFromTarget
verifyCallResult
isContract
revert
target
staticcall
delegatecall
Initializable
disableInitializers
type
initializer
onlyInitializing
initializer
_getInitializedVersion
_isInitializing
AddressUpgradeable (lib)
sendValue
functionCall
functionStaticCall
functionCallWithValue
functionDelegateCall
verifyCallResultFromTarget
verifyCallResult
revert
isContract
ERC1967Upgrade (Upgradeable)
__ERC1967Upgrade_init
__ERC1967Upgrade_init_unchained
_upgradeToAndCall
_upgradeToAndCallUUPS
_setRollback
_upgradeTo
_setImplementation
getAdmin
_changeAdmin
_setAdmin
_getImplementation
getBeacon
StorageSlotUpgradeable (lib)
getAddressSlot
getBooleanSlot
getBytes32Slot
getUint256Slot
getStringSlot
getBytesSlot

Summary

Sphere contract implements a staking, and rewards mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io