Cyberscope

Audit Report

Origin Protocol Staking

December 2022

Type	ERC20
Network	ETH
Address	0x3675c3521f8a6876c8287e9bb51e056862d1399b
Audited by	© cyberscope

Table of Contents

Review

Contract Name	SingleAssetStaking
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	https://etherscan.io/address/0x3675c3521f8a6876c8287e9bb51e056862d1399b
Address	0x3675c3521f8a6876c8287e9bb51e056862d1399b
Network	ETH

Audit Updates

Initial Audit	19 Dec 2022

Source Files

Filename	SHA256
@openzeppelin/contracts/token/ERC20/IERC20.sol	6eacf8ca56b41b7636489c0996d8f9608 b4d298879a1fd5d876f21ad7a6711f1
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	b5a1340c5232f387b15592574f27eef78f 6017bdc66542a1cea512ad4f78a0d2
@openzeppelin/contracts/utils/Address.sol	0228dd7c0a0d1342b88eab6e5a4a07ae 4350818ba1650be0a374064b02218f37
@openzeppelin/contracts/utils/math/SafeMath.sol	15941f3904992a62ed117e93d9e2d5c4c 22bd09a7ff97fdd5f49273cf09703ac
contracts/governance/Governable.sol	dd402874d61786fc4ccc83c5cf24454e2 1ff2a3d614bfa8c20fcdb7c798455b4
contracts/staking/SingleAssetStaking.sol	d524d97dabfb7d46da56169173e354a13 6d3a22f3609d89d838cf205ca2571e4
contracts/utils/Initializable.sol	64846219cfd26b59d19137f2600d97967 01502fab42d5c7d44a516f8fb1399f8
contracts/utils/StableMath.sol	a3351d074530ed3a916ff185b1f1b350bf 14653953a741ea19edc4bd40877c09

Introduction

The SingleAssetStaking contract implements a staking mechanism. The contract prodives 3 different ways to stake tokens.

stake()

Any user has the ability to stake any amount of tokens for a specific duration and rate. The staked amount is transferred from the user's address to the staking contract.

stakeWithSender()

The contract of the staking token has the ability to stake any amount of tokens for a specific duration and rate. The staked amount is transferred from the user's address to the staking contract via the staking token contract.

airDroppedStake()

Some users that are choosed by the staking contract admins have the ability to stake tokens for a predefined amount, duration and rate. The staked amount is not transferred but is provided by the admins as a voucher.

Rate and Duration

The rate and duration for the stake() and stakeWithSender() methods are enumerated. The airDroppedStake() method does not validate the rate and duration, it is based on the off-chain validation.

Reward Amount Reserves

The reward amount is guaranteed to be available. The staking contract checks if the reserves are sufficient to cover the amount on every stake operation.

Airdrop Stake Merkle Proof Mechanism

The staking contract uses a Merkle Proof mechanism in order to configure the airdrop applicable addresses. The verification process is based on an off-chain configuration. The contract owner is responsible for updating the in-chain "Merkle Root" in order to validate correctly the provided message.

According to the Markle algorithm, the off-chain mechanism pre-defines all the applicable users by:

- The stake type
- Sender's address
- Staking Duration
- Staking Rate
- Staking amount

We state that the Merkle Proof algorithm is required for proper protocol operations and gas consumption decrease. Thus, we emphasise that the Merkle proof algorithm is based on an off-chain mechanism. Any off-chain mechanism could potentially be compromised and affect the on-chain state unexpectedly. Hence we emphasise the Governor's role to be extra careful with the credentials.

Roles

The Governor role has the authority to

- Pause the transactions
- Set the staking rates and durations
- Set the Merkle Proof roots per type
- Transfer all the user’s stakes. This requires the approval of the user via signature.

The Staking contract has the authority to

- Stake user’s tokens

Any user has the authority to

- Stake
- Unstake (Exit)

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	ADTS	Appropriate Data Type Size	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

ADTS - Appropriate Data Type Size

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L47
Status	Unresolved

Description

The contract is using the uint256 data type in order to store the `MAX_STAKES` variable. The `MAX_STAKES` variable is constant to the `256` number. Hence, `2^8 + 1` digits are used.

```
uint256 constant MAX_STAKES = 256;
```

Additionally, the contract uses a `uint240` data type in order to store the `rate` in the Stake structure. The `rate` variable is validated by the `uint256[] public rates;` variable that is represented by a `uint256` data type. As a result, the maximum acceptable number that the `rates` array can accept is `2^240` digits.

```
uint240 rate;
...
uint256[] public rates;
```

Recommendation

Since the value `256` can be represented by `2^8 + 1` digits, the team is advised to use a `uint16` data type, which will use fewer bits and take up less storage space on the blockchain. In the same manner, the contract could use a `uint240` data type in order to store the `rates` array.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L96,117,269,299
Status	Unresolved

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.

```
function totalStaked(address account)
    external
    view
    returns (uint256 total)
{
    Stake[] storage stakes = userStakes[account];
...

function _airDroppedStakeClaimed(address account, uint8 stakeType)
    internal
    view
    returns (bool)
{
    Stake[] storage stakes = userStakes[account];

...
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L5
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows. Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases unnecessary the gas consumption.

```
using SafeMath for uint256;
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result. If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement. Read more about the breaking change on
https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L207
Status	Unresolved

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 they transfer the token 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 behaviour.

```
stakingToken.safeTransferFrom(staker, address(this), amount);
```

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

Recommendation

The team is advised to take into consideration the actual amount that has 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 should take in consideration the potential deducted amount after the transfer.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L104,278,441
Status	Unresolved

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

```
total = total.add(stake.amount.mulTruncate(stake.rate));
total = total.add(stakes[i].amount);
totalWithdraw = totalWithdraw.add(_totalExpected(exitStake));
stakedAmount = stakedAmount.add(exitStake.amount);
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L62,63,64,245,463,464,498,509,510,519,530,531,532
Status	Unresolved

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 your Solidity code, making it easier for others to understand and work with.

The followings are 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 behaviour of your code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _stakingToken
uint256[] calldata _durations
uint256[] calldata _rates
uint256 _duration
address _frmAccount
address _dstAccount
bool _paused
uint256[] calldata _durations
uint256[] calldata _rates
address _agent
uint8 _stakeType
bytes32 _rootHash
uint256 _proofDepth
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

You can find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L102,109,307
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
Stake storage stake = stakes[i]
Stake storage _stake
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L520
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
transferAgent = _agent
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/staking/SingleAssetStaking.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows you to specify a minimum version of the Solidity compiler that must be used to compile your contract code. This is useful because it allows you to ensure that your contract will be compiled using a version of the compiler that is known to be compatible with your code.

```
pragma solidity ^0.8.0;
```

Recommendation

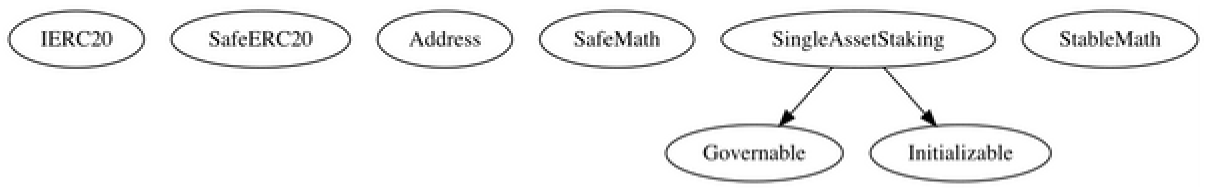
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Governable	Implementation			
		Public	✓	-
	governor	Public		-
	_governor	Internal		
	_pendingGovernor	Internal		
	isGovernor	Public		-
	_setGovernor	Internal	✓	
	_setPendingGovernor	Internal	✓	
	transferGovernance	External	✓	onlyGovernor
	claimGovernance	External	✓	-
	_changeGovernor	Internal	✓	
SingleAssetStaking	Implementation	Initializable, Governable		
	initialize	External	✓	onlyGovernor initializer
	_setDurationRates	Internal	✓	
	_totalExpectedRewards	Internal		
	_totalExpected	Internal		
	_airDroppedStakeClaimed	Internal		
	_findDurationRate	Internal		
	_stake	Internal	✓	
	_stakeWithChecks	Internal	✓	
	getAllDurations	External		-
	getAllRates	External		-

	getAllStakes	External		-
	durationRewardRate	External		-
	airDroppedStakeClaimed	External		-
	totalStaked	External		-
	totalExpectedRewards	External		-
	totalCurrentHoldings	External		-
	airDroppedStake	External	✓	requireLiquidity
	stake	External	✓	requireLiquidity
	stakeWithSender	External	✓	requireLiquidity
	exit	External	✓	requireLiquidity
	transferStakes	External	✓	-
	setPaused	External	✓	onlyGovernor
	setDurationRates	External	✓	onlyGovernor
	setTransferAgent	External	✓	onlyGovernor
	setAirDropRoot	External	✓	onlyGovernor
Initializable	Implementation			
StableMath	Library			
	scaleBy	Internal		
	mulTruncate	Internal		
	mulTruncateScale	Internal		
	mulTruncateCeil	Internal		
	divPrecisely	Internal		

Inheritance Graph



Flow Graph

Summary

Origin Protocol Staking contract implements a staking mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

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