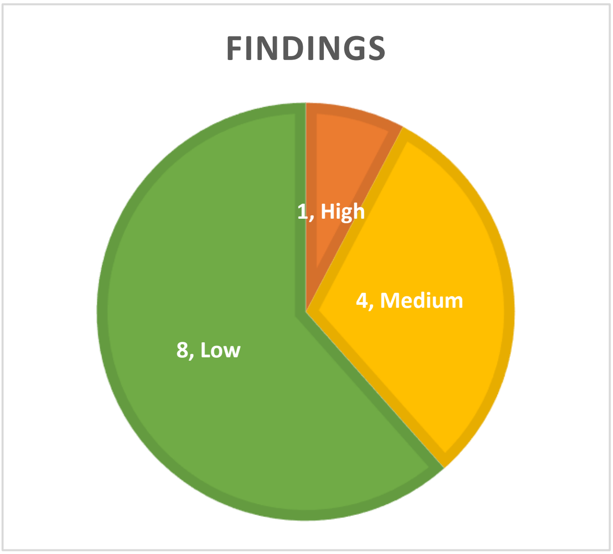
Trust Security

Smart Contract Audit

BugHole Restaking

05/11/24

Executive summary

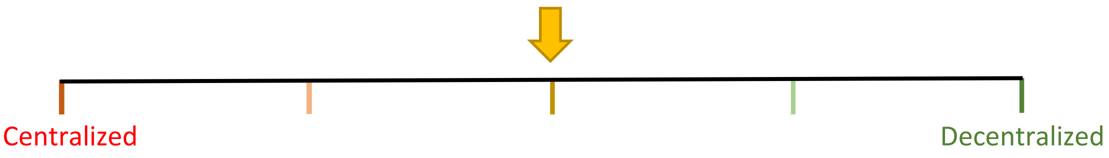


Category	Liquid Staking
Audited file count	14
Lines of Code	812
Auditor	cccz 0xTheC0der
Time period	28/10/2024- 01/11/2024

Findings

Severity	Total	Fixed	Open	Acknowledged
High	1	1	-	-
Medium	4	4	-	-
Low	8	6	-	2

Centralization score



Signature

Digitally signed by Trust
Date: 2024.11.05 18:38:05 +02'00'

EXECUTIVE SUMMARY 1

DOCUMENT PROPERTIES 4

Versioning **4**

Contact **4**

INTRODUCTION 5

Scope **5**

Repository details **5**

About Trust Security **5**

About the Auditors **5**

Disclaimer **6**

Methodology **6**

QUALITATIVE ANALYSIS 7

FINDINGS 8

High severity findings **8**
TRST-H-1 Unauthenticated attacker can burn anyone's ReStakingToken 8

Medium severity findings **9**
TRST-M-1 User cannot claim the last entry of the unstakeList 9
TRST-M-2 ReStakingToken.setReStakingManagerAddress() has incorrect role requirements 9
TRST-M-3 The sKlay to Kaia ratio does not factor in rewards 10
TRST-M-4 The latest ratio of stKlay is not used when transferring stKlay 11

Low severity findings **14**
TRST-L-1 Implicit msg.sender restriction at initialization 14
TRST-L-2 Some functions don't use the whenNotPaused modifier but should 14
TRST-L-3 Single-step ownership transfer is error-prone 15
TRST-L-4 Contracts accept native Kaia but have no way to retrieve it 15
TRST-L-5 Allowance should not be required for burning ReStakingToken 16
TRST-L-6 Missing zero address checks is error prone 16
TRST-L-7 approveToken() may not set allowedTotalAmount 17
TRST-L-8 Updating claimIntervalTime affects user claiming 18

Additional recommendations **20**
TRST-R-1 OwnableUpgradeable can be removed in some contracts 20
TRST-R-2 ReentrancyGuardUpgradeable can be removed in some contracts 20
TRST-R-3 Create base contract for ReStaking* contracts 20
TRST-R-4 The addRole() and removeRole() methods can be removed 20
TRST-R-5 Add None state to UnstakeState enum 20

TRST-R-6 RE_STAKING_MANAGER role can be removed in some contracts 20
TRST-R-7 Modifier onlyStakeToken() should be used or removed 21
TRST-R-8 Modifier onlyReStakingManager() should be used or removed 21
TRST-R-9 Avoid hard-coded constants in code 21
TRST-R-10 Remove isApproveCheck and related unused code paths 21
TRST-R-11 Use safeTransferFrom() instead of transferFrom() 21
TRST-R-12 Remove SafeMath 21
TRST-R-13 Fix typos 22
TRST-R-14 Incorrect error message may be emitted when allowedTotalAmount is not enough 22
TRST-R-15 Unstake does not use the latest stKaia ratio 22
TRST-R-16 Ineffective role segregation at initialization 22
TRST-R-17 approveToken() accepts unsupported tokens 22
TRST-R-18 The exchange() function does not verify incoming tokens 23
TRST-R-19 The same bufferTokenAmount applies different tokens 23
TRST-R-20 The admin account can be permanently revoked 24
TRST-R-21 calculateStakingToken() returns zero on unsupported token 24

Centralization risks 25
TRST-CR-1 RE_STAKING_MANAGER can mint ReStakingToken at will 25
TRST-CR-2 The admin controls various crucial setters across the protocol 25

Systemic risks 26
TRST-SR-1 stKaia solvency of *ReStakingTransfer* contract is dependent on external actions 26

Document properties

Versioning

Version	Date	Description
0.1	01/11/2024	Client report
0.2	05/11/2024	Mitigation review
0.3	05/11/2024	Mitigation review #2

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

- contracts/restaking/ReStakingManager/ReStakingManager.sol
- contracts/restaking/interface/ICnStakingV2.sol
- contracts/restaking/Transfer/ReStakingTransfer.sol
- contracts/restaking/Treasury/ReStakingTreasury.sol
- contracts/restaking/ReStakingToken/ReStakingToken.sol
- contracts/restaking/ReStakingManager/IReStakingManager.sol
- contracts/restaking/interface/IStakingToken.sol
- contracts/restaking/library/Validator.sol
- contracts/restaking/Transfer/IReStakingTransfer.sol
- contracts/restaking/ReStakingToken/IReStakingToken.sol
- contracts/restaking/Treasury/IReStakingTreasury.sol
- contracts/restaking/enums/State.sol
- contracts/restaking/structs/Token.sol
- contracts/restaking/structs/Unstake.sol

Repository details

- **Repository URL:** https://github.com/bug4city/bughole-lsd/
- **Commit hash:** f3d6a0153ebfcaff6088d1f6aa210706ad40b6ea
- **Mitigation review hash:** 84fb908daa9b8f8d6065168cbb436d8dc5a09b14
- **Mitigation review hash #2:** 82e27bfc9ac63c2c25b638a8eab55c43dbb64f03

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

About the Auditors

A top competitor in audit contests, cccz has achieved superstar status in the security space. He is a Black Hat / DEFCON speaker with rich experience in both traditional and blockchain security.

0xTheC0der is a smart contract security researcher with a passion for physics. He got into the Web3 space via bug bounties and audit contests with multiple top finishes. In addition to being an audit contest judge, he has reviewed 25+ blockchain protocols, mostly EVM based but also on Substrate, NEAR, Solana and Starknet.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	**Rating**	**Comments**
Code complexity	**Good**	Project kept code as simple as possible, reducing attack risks
Documentation	**Good**	Project is mostly very well documented.
Best practices	**Good**	Project consistently adheres to industry standards.
Centralization risks	**Moderate**	Project relies on admin to set correct parameters. A compromised admin account could risk the safety of funds.

Findings

High severity findings

TRST-H-1 Unauthenticated attacker can burn anyone's ReStakingToken

- **Category:** Access-control issues
- **Source:** ReStakingToken.sol
- **Status:** Fixed

Description

The *burn()* method is permissionless. Since no allowance is required to burn ERC-20 tokens, any user can burn anyone's *ReStakingToken* leading to a full loss and inability to correctly unstake.

Recommended mitigation

It is recommended to add the **onlyRole(RE_STAKING_MANAGER)** modifier similar to the *mint()* method.

```
-   function burn(address account, uint256 amount) public {
+   function burn(address account, uint256 amount) public onlyRole(RE_STAKING_MANAGER) {
        _burn(account, amount);
    }
```

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

Medium severity findings

TRST-M-1 User cannot claim the last entry of the unstakeList

- **Category:** Validation issues
- **Source:** ReStakingManager.sol
- **Status:** Fixed

Description

When unstaking, a corresponding entry is added to *unstakeList[sender]*, which allows to claim stKaia after a given claim period.

However, there is the following check in the *claim()* and *_claim()* functions.

```
require(unstakeList[sender].length - 1 > index, "ReStakingManager:: index is not
exist");
```

Consider the scenario where a user is unstaking once, i.e. **unstakeList[sender].length == 1** and **index == 0**.

Then, the above condition will evaluate to **0 > 0** causing the check to fail, which prevents claiming of stKaia while the corresponding *ReStakingToken* were already burned.

Recommended mitigation

It is recommended to modify the checks as follows.

```
-   require(unstakeList[sender].length - 1 > index, "ReStakingManager:: index is not
exist");
+   require(unstakeList[sender].length > index, "ReStakingManager:: index is not
exist");
```

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-M-2 ReStakingToken.setReStakingManagerAddress() has incorrect role requirements

- **Category:** Authentication issues
- **Source:** ReStakingToken.sol
- **Status:** Fixed

Description

ReStakingToken.setReStakingManagerAddress() is only accessible for the **RE_STAKING_MANAGER** role. However, the underlying calls to *removeRole()* and *addRole()* require the **DEFAULT_ADMIN_ROLE**.

Consequently, invoking this method will always fail except when the **RE_STAKING_MANAGER** role and **DEFAULT_ADMIN_ROLE** are associated with the same account.

```
    function setReStakingManagerAddress(address payable _reStakingManagerAddress)
public onlyRole(RE_STAKING_MANAGER) {
        require(_reStakingManagerAddress != address(0), "ReStakingManager:: zero
address");
        require(_reStakingManagerAddress != reStakingManagerAddress,
"ReStakingManager:: same address");

        removeRole(RE_STAKING_MANAGER, reStakingManagerAddress);
        reStakingManagerAddress = _reStakingManagerAddress;
        addRole(RE_STAKING_MANAGER, reStakingManagerAddress);
    }
...
    function addRole(bytes32 role, address account) public
onlyRole(DEFAULT_ADMIN_ROLE) {
        _grantRole(role, account);
    }
    function removeRole(bytes32 role, address account) public
onlyRole(DEFAULT_ADMIN_ROLE) {
        _revokeRole(role, account);
    }
```

Recommended mitigation

Since it doesn't make sense to allow the **RE_STAKING_MANAGER** role to add/remove **RE_STAKING_MANAGER** roles, it's recommended to allow **DEFAULT_ADMIN_ROLE** to call *ReStakingToken.setReStakingManagerAddress().*

```
-   function setReStakingManagerAddress(address payable _reStakingManagerAddress)
public onlyRole(RE_STAKING_MANAGER) {
+   function setReStakingManagerAddress(address payable _reStakingManagerAddress)
public onlyRole(DEFAULT_ADMIN_ROLE) {
```

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-M-3 The sKlay to Kaia ratio does not factor in rewards

- **Category:** Accounting issues
- **Source:** ReStakingManager.sol
- **Status:** Fixed

Description

For sKlay, staking Kaia to *cnStakingV2* will generate rewards, and the rewards will be distributed to *Reward: Ozys 3*, these rewards are not immediately staked to *cnStakingV2*.

In *ReStakingManager*, it uses the following code to calculate the ratio of sKlay to Kaia.

```
    uint256 sKlayRatio = (cnStakingV2.staking() - cnStakingV2.unstaking()) * 10 ** 18
/ sKLAY.totalSupply();
```

```
    uint256 nativeTokenAmount = tokenAmount * sKlayRatio / 10 ** 18;
```

The problem here is that it doesn't factor in rewards in *Reward: Ozys 3*.

Consider staking 1000 Kaia and minting 1000 sKlay, and then generating 100 Kaia to *Reward: Ozys 3*, so the current Kaia:sKlay ratio should be 1.1, but since **staking() - unstaking()** does not include the 100 Kaia rewards, the **sKlayRatio** calculated in *ReStakingManager* is 1, this will cause stakers to lose assets.

Recommended mitigation

It is recommended to call *refreshStaking()* of *KLAYstation: Ozys V2* to stake rewards into *cnStakingV2* before calculating **sKlayRatio**.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-M-4 The latest ratio of stKlay is not used when transferring stKlay

- **Category:** Accounting issues
- **Source:** ReStakingManager.sol
- **Status:** Fixed

Description

stKlay is a rebase token that mints a fixed amount of shares to the user when the user stakes Kaia, and the user's stKlay balance will be the product of the share amount and the ratio.

```
    function balanceOf(address user)
        public
        view
        override(IKIP7, KIP7Upgradeable)
        returns (uint256)
    {
        return _getKlayByShares(sharesOf(user));
    }
...
    function _getKlayByShares(uint256 sharesAmount)
        private
        view
        returns (uint256)
    {
        return (sharesAmount * stakingSharesRatio) / 10**27;
    }
```

When staking, stKlay will call *nodeManager.distributeReward()* to update the ratio.

```
    function stake() external payable whenNotPaused {
        nodeManager.distributeReward();
        _stake(_msgSender(), msg.value);
}
...
```

```
    function _distributeReward() private {
        uint256 rewards = address(this).balance;

        if (rewards < distributeMinimum) return;

        uint256 fee = (rewards * feeRate) / 1e4;
        uint256 reStaking = rewards - fee;

        stKlay.increaseTotalStaking(reStaking);
...
    function increaseTotalStaking(uint256 amount)
        external
        onlyNodeManager
        whenNotPaused
        nonZero(amount)
        nonReentrant
    {
        totalRestaked += amount;
        totalStaking += amount;
        if (totalShares > 0)
            stakingSharesRatio = (totalStaking * 10**27) / totalShares;
        emit RestakedFromManager(totalRestaked, amount, totalStaking);
    }
```

There is a problem with stKlay. When transferring tokens, *nodeManager.distributeReward()* is not called to update the ratio.

```
    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual override returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, amount);
        return true;
    }
...
    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override whenNotPaused {
        _beforeTokenTransfer(from, to, amount);

        _transferShares(from, to, _getSharesByKlay(amount));
        emit Transfer(from, to, amount);

        _afterTokenTransfer(from, to, amount);
    }
...
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
...
    function _getSharesByKlay(uint256 klayAmount)
        private
        view
        returns (uint256)
    {
        return
            (klayAmount * 10**27 + stakingSharesRatio - 1) / stakingSharesRatio;
    }
```

Considering that the stale ratio is 1.0 and the latest ratio is 1.1, when a user stake 1100 stKlay, 1100 shares are transferred, however, according to the latest ratio, only 1000 shares need to be transferred.

Recommended mitigation

It is recommended to call stKlay's *nodeManager.distributeReward()* to update stKlay's ratio when **tokenAddress == stKlay**.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

Low severity findings

TRST-L-1 Implicit msg.sender restriction at initialization

- **Category:** Authentication issues
- **Source:** ReStakingTreasury.sol, ReStakingTransfer.sol, ReStakingToken.sol, ReStakingManager.sol
- **Status:** Fixed

Description

In the *initialize()* methods of *ReStakingTreasury*, *ReStakingTransfer*, *ReStakingToken*, *ReStakingManager* contracts, there are subsequent calls that require the **DEFAULT_ADMIN_ROLE**. However, this role is granted to the **_owner** account. Therefore, it is implicitly required that **msg.sender == _owner**, otherwise these subsequent calls will fail.

```
_grantRole(DEFAULT_ADMIN_ROLE, _owner);
...

// caller is msg.sender but requires DEFAULT_ADMIN_ROLE
setReStakingManagerAddress(_reStakingManagerAddress);
setStakingTokenAddress(_stakingTokenAddress);
setReStarkingTreasuryAddress(_reStarkingTreasuryAddress);
```

Recommended mitigation

It is recommended to temporarily grant the **DEFAULT_ADMIN_ROLE** to **msg.sender** on initialization to process the permissioned calls.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-L-2 Some functions don't use the whenNotPaused modifier but should

- **Category:** Integration issues
- **Source:** ReStakingTreasury.sol, ReStakingTransfer.sol, ReStakingManager.sol
- **Status:** Fixed

Description

The four contracts *ReStakingToken*, *ReStakingTransfer*, *ReStakingTreasury*, and *ReStakingManager* all inherit the *PausableUpgradeable* contract, but only *ReStakingToken* uses the *whenNotPaused* modifier.

This will only allow the upstream and downstream of *ReStakingToken.approve()* and *ReStakingToken._update()* to be paused, while other functions such as *claim()* cannot be paused.

Recommended mitigation

It is recommended to apply the *whenNotPaused* modifier to the functions in *ReStakingTransfer*, *ReStakingTreasury*, and *ReStakingManager* contracts.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-L-3 Single-step ownership transfer is error-prone

- **Category:** Logical issues
- **Source:** ReStakingTransfer.sol
- **Status:** Fixed

Description

The base *OwnableUpgradeable* contract's *renounceOwnership()/transferOwnership()* methods allow the owner to give up ownership in a single step which can be detrimental for the functionality of the protocol, specifically when it comes to calling the *withdraw()* method.

Recommended mitigation

It is recommended to rely on a two-step ownership transfer pattern which is much less error-prone.

Team response

Fixed.

Mitigation Review

Two-step ownership transfer is implemented. However, there are the public base *OwnableUpgradeable* contract methods *renounceOwnership()* and *transferOwnership()*, which need to be overridden to prevent one-step transfer or self-renouncing.

Team response #2

Fixed.

Mitigation Review #2

The fix implements the recommendation.

TRST-L-4 Contracts accept native Kaia but have no way to retrieve it

- **Category:** Logical issues
- **Source:** ReStakingTreasury.sol, ReStakingTransfer.sol, ReStakingToken.sol, ReStakingManager.sol
- **Status:** Acknowledged

Description

ReStakingTreasury, *ReStakingTransfer*, *ReStakingToken*, *ReStakingManager* contracts implement a method to receive native Kaia.

```
receive() external payable {}
```

However, there is no corresponding method that allows to retrieve it again, therefore any native funds transferred to these contracts will be lost.

Recommended mitigation

It is recommended to remove the *receive()* method if the contract is not intended to receive native Kaia.

Team response

Acknowledged.

TRST-L-5 Allowance should not be required for burning ReStakingToken

- **Category:** Validation issues
- **Source:** ReStakingManager.sol
- **Status:** Fixed

Description

The *_unstake()* method, which subsequently burns the caller's *ReStakingToken*, employs an allowance check.

```
require(IReStakingToken(getReStakingTokenAddress()).allowance(sender, address(this)) >=
reStakingTokenAmount, "ReStakingManager:: allowance is not enough");
```

However, allowance is not required for burning ERC-20 tokens, therefore this check is superfluous and impedes UX by requiring an additional approval.

Recommended mitigation

It is recommended to remove the allowance check.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-L-6 Missing zero address checks is error prone

- **Category:** Validation issues
- **Source:** ReStakingTreasury.sol, ReStakingTransfer.sol, ReStakingToken.sol, ReStakingManager.sol
- **Status:** Fixed

Description

Various setter methods across the protocol enable the admin to set addresses which are crucial for the functionality of the protocol. However, in these setters it is not assured that the address is not set to **address(0)**.

Recommended mitigation

It is recommended to implement the aforementioned address zero checks.

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-L-7 approveToken() may not set allowedTotalAmount

- **Category:** Logical issues
- **Source:** ReStakingManager.sol
- **Status:** Fixed

Description

When first approving the token, the **state** and **allowedTotalAmount** will be set.

```
        if (!isExist) {
            approvedTokenList[tokenAddress] = Token.TokenInfo(approvedTokenArray.length,
tokenAddress, State.ApprovedTokenState.Approved, allowedTotalAmount, 0, 0);
            approvedTokenArray.push(approvedTokenList[tokenAddress]);
```

When disapproving the token, the **state** is changed and **allowedTotalAmount** is set to 0.

```
    function disapproveToken(address tokenAddress) public onlyRole(DEFAULT_ADMIN_ROLE)
validAddress(tokenAddress) {
        require(approvedTokenList[tokenAddress].state ==
State.ApprovedTokenState.Approved, "ReStakingManager:: not approved token");

        approvedTokenList[tokenAddress].state = State.ApprovedTokenState.Removed;
        approvedTokenArray[approvedTokenList[tokenAddress].index].state =
approvedTokenList[tokenAddress].state;

        approvedTokenList[tokenAddress].allowedTotalAmount = 0;
        approvedTokenArray[approvedTokenList[tokenAddress].index].allowedTotalAmount =
approvedTokenList[tokenAddress].allowedTotalAmount;
    }
```

The problem here is that when the token is re-approved, only the **state** is changed, not the **allowedTotalAmount**.

```
        } else {
            approvedTokenList[tokenAddress].state = State.ApprovedTokenState.Approved;
            approvedTokenArray[approvedTokenList[tokenAddress].index].state =
approvedTokenList[tokenAddress].state;
```

```
        }
```

Recommended mitigation

It is recommended to change to:

```
        } else {
            approvedTokenList[tokenAddress].state = State.ApprovedTokenState.Approved;
            approvedTokenArray[approvedTokenList[tokenAddress].index].state = 
approvedTokenList[tokenAddress].state;
+           approvedTokenList[tokenAddress].allowedTotalAmount = allowedTotalAmount;
+           approvedTokenArray[approvedTokenList[tokenAddress].index].allowedTotalAmount 
= approvedTokenList[tokenAddress].allowedTotalAmount;
        }
```

Team response

Fixed.

Mitigation Review

The fix implements the recommendation.

TRST-L-8 Updating claimIntervalTime affects user claiming

- **Category:** Logical issues
- **Source:** ReStakingManager.sol
- **Status:** Acknowledged

Description

Admin can set **claimIntervalTime**, which will affect the user's claim.

```
    function setClaimIntervalTime(uint256 _claimIntervalTime) public 
onlyRole(DEFAULT_ADMIN_ROLE) {
        require(_claimIntervalTime > 0, "ReStakingManager:: claimIntervalTime is zero");

        claimIntervalTime = _claimIntervalTime;
    }
...
    function getClaimIntervalTime() public view returns (uint256) {
        return claimIntervalTime;
    }
...
        require(block.timestamp >= unstakeList[sender][index].unstakeTime + 
getClaimIntervalTime(), "ReStakingManager:: claimIntervalTime is not passed");
```

Consider that the **claimIntervalTime** is 3 days, and the user initiates unstake and expects to claim after 3 days.

The admin changes the **claimIntervalTime** to 7 days, and the user can only claim after 7 days, which breaks the user's expectations.

Recommended mitigation

It is recommended to use the current **claimIntervalTime** to determine the time they can claim when the user unstakes.

Team response

Acknowledged**.**

Additional recommendations

TRST-R-1 OwnableUpgradeable can be removed in some contracts

OwnableUpgradeable is unused in *ReStakingTreasury*, *ReStakingToken* and *ReStakingManager,* so it can be removed.

TRST-R-2 ReentrancyGuardUpgradeable can be removed in some contracts

ReentrancyGuardUpgradeable is unused in *ReStakingToken,* so it can be removed.

TRST-R-3 Create base contract for ReStaking* contracts

The *ReStakingTreasury*, *ReStakingTransfer*, *ReStakingManager* and *ReStakingToken* all inherit from *OwnableUpgradeable*, *AccessControlUpgradeable*, etc. and implement the same methods like *addRole(), pause()*, etc.

Therefore, it is beneficial to create a common base contract that encapsules this shared functionality.

TRST-R-4 The addRole() and removeRole() methods can be removed

In *AccessControlUpgradeable* (protocol is using OpenZeppelin v5.0.2) there already exist public *grantRole()* and *revokeRole()* methods.

TRST-R-5 Add None state to UnstakeState enum

The default **UnstakeState** is **Unstaked**, which indicates that the **UnstakeState** may be valid, it is recommended to add **None** as the default state to distinguish between uninitialized **UnstakeState** and valid **UnstakeState**.

```
-   enum UnstakeState { Unstaked, Claimed }
+   enum UnstakeState { None, Unstaked, Claimed }
```

TRST-R-6 RE_STAKING_MANAGER role can be removed in some contracts

RE_STAKING_MANAGER role is unused for access control in *ReStakingTreasury*, *ReStakingTransfer*, *ReStakingManager*, so it can be removed.

TRST-R-7 Modifier onlyStakeToken() should be used or removed

onlyStakeToken() modifier is unused in *ReStakingManager*, so it should be used or removed.

TRST-R-8 Modifier onlyReStakingManager() should be used or removed

onlyReStakingManager() modifier is unused in *ReStakingToken*, so it should be used or removed.

TRST-R-9 Avoid hard-coded constants in code

Avoid using hard-coded constants in code.

```
gcKlayAddress = address(0x999999999939Ba65AbB254339eEc0b2A0daC80E9);
stKlayAddress = address(0xF80F2b22932fCEC6189b9153aA18662b15CC9C00);
sKlayAddress = address(0xA323d7386b671E8799dcA3582D6658FdcDcD940A);
sKlayGCAddress = payable(0x06cD16588aE09DEEa859Dcfc8b67386Bca412433);

...

approveToken(_stakingTokenAddress, 500000000000000000000000000);
```

TRST-R-10 Remove isApproveCheck and related unused code paths

The *_stake()* method in *ReStakingManager* has a parameter bool **isApproveCheck**. In the current implementation, *_stake()* is only called with **isApproveCheck == true**. Therefore, the parameter and all code paths related to **isApproveCheck == false** can be removed.

TRST-R-11 Use safeTransferFrom() instead of transferFrom()

In *ReStakingManager*, there are 2 instances where ERC-20 *transferFrom()* is used. It is recommended to instead rely on *safeTransferFrom()* in these instances. Note that this does not conflict with *KIP7.safeTranferFrom()*.

TRST-R-12 Remove SafeMath

Solidity versions >= 0.8.0 already comes with built-in overflow checking.

TRST-R-13 Fix typos

There are multiple instances across the protocol where “**Starking”** is written instead of “**Staking”**.

TRST-R-14 Incorrect error message may be emitted when allowedTotalAmount is not enough

When **allowedTotalAmount** is not enough, the calculation of **allowedTotalAmount - totalExchangeAmount** may revert first due to underflow instead of throwing the error message. It is recommended to change to the following.

```
-       require((approvedTokenList[tokenAddress].allowedTotalAmount -
approvedTokenList[tokenAddress].totalExchangeAmount + bufferTokenAmount) >= amount,
"ReStakingManager:: allowedTotalAmount is not enough");
+       require((approvedTokenList[tokenAddress].allowedTotalAmount + bufferTokenAmount)
>= amount + approvedTokenList[tokenAddress].totalExchangeAmount, "ReStakingManager::
allowedTotalAmount is not enough");
```

TRST-R-15 Unstake does not use the latest stKaia ratio

unstake() calls *getRatioNativeTokenByStakingToken()* to get the Kaia amount corresponding to the stKaia, but since *NodeController.distributeReward()* is not called to update the ratio before that, this will result in the conversion using the stale ratio.

```
        uint256 nativeTokenAmount =
IStakingToken(getStakingTokenAddress()).getRatioNativeTokenByStakingToken(reStakingToken
Amount);
```

TRST-R-16 Ineffective role segregation at initialization

In *ReStakingTreasury*, *ReStakingTransfer*, *ReStakingToken*, *ReStakingManager* contracts, different roles are initially assigned to the same **_owner** account, it is recommended to assign different roles to different accounts already at initialization.

```
_grantRole(DEFAULT_ADMIN_ROLE, _owner);
_grantRole(RE_STAKING_PAUSE, _owner);
_grantRole(RE_STAKING_MANAGER, _owner);
```

TRST-R-17 approveToken() accepts unsupported tokens

ReStakingManager.approveToken() methods suggests that arbitrary tokens can be approved for re-staking. However, the actually supported tokens are given by the *calculateStakingToken()* method. They are stKaia, gcKlay, stKlay and sKlay. It is recommended to reconsider the *approveToken()* concept.

```
        if (tokenAddress == getStakingTokenAddress()) {
            return tokenAmount;
        } else if (tokenAddress == gcKlayAddress) { // gcKLAY = Kaia 1:1
            return
IStakingToken(getStakingTokenAddress()).getRatioStakingTokenByNativeToken(tokenAmount);
        } else if (tokenAddress == stKlayAddress) { // stKLAY = Kaia 1:1
            return
IStakingToken(getStakingTokenAddress()).getRatioStakingTokenByNativeToken(tokenAmount);
        } else if (tokenAddress == sKlayAddress) { // sKLAY = Kaia 1:n
            ICnStakingV2 cnStakingV2 = ICnStakingV2(sKlayGCAddress);
            IERC20 sKLAY = IERC20(tokenAddress);

            uint256 sKlayRatio = (cnStakingV2.staking() - cnStakingV2.unstaking()) * 10
** 18 / sKLAY.totalSupply();
            uint256 nativeTokenAmount = tokenAmount * sKlayRatio / 10 ** 18;

            return
IStakingToken(getStakingTokenAddress()).getRatioStakingTokenByNativeToken(nativeTokenAmo
unt);
        } else {
            revert("ReStakingManager:: not supported token");
        }
```

TRST-R-18 The exchange() function does not verify incoming tokens

The *exchange()* method trusts the *ReStakingManager* contract that the specified **tokenAddress/tokenAmount** was actually transferred to the *ReStakingTransfer* contract.

It is recommended to implement a pull-pattern where the *exchange()* method transfers the tokens in itself.

TRST-R-19 The same bufferTokenAmount applies different tokens

This is the current usage of **bufferTokenAmount** in the *_stake()* method.

```
require((approvedTokenList[tokenAddress].allowedTotalAmount -
approvedTokenList[tokenAddress].totalExchangeAmount + bufferTokenAmount) >= amount,
"ReStakingManager:: allowedTotalAmount is not enough");
```

The **bufferTokenAmount** is a universal value that is applied for each approved token. However, these tokens might not have the same decimals as **bufferTokenAmount** allowing to exceed the **allowedTotalAmount** by unexpectedly high/low amounts. It is recommended to make **bufferTokenAmount** a percentage of **allowedTotalAmount** instead of a universal value.

TRST-R-20 The admin account can be permanently revoked

The *removeRole()* method and the base AccessControlUpgradeable contract's *revokeRole()/renounceRole()* methods allow the admin to self-revoke the **DEFAULT_ADMIN_ROLE**, which can be detrimental for the functionality of the protocol.

It is recommended to implement a modifier that prevents the above scenario and to override the base contract's methods accordingly.

TRST-R-21 calculateStakingToken() returns zero on unsupported token

The *calculateStakingToken()* method accepts any admin-approved token, but only implements support for stKaia, gcKlay, stKlay and sKlay. In case of any other token, the method will return zero, which is a silent failure that might pose a risk when integrating this method.

It is recommended to prevent usage and approval of tokens that are unsupported on contract level.

Centralization risks

TRST-CR-1 RE_STAKING_MANAGER can mint ReStakingToken at will

Admin can grant any account the **RE_STAKING_MANAGER** role and mint ReStakingToken at will.

TRST-CR-2 The admin controls various crucial setters across the protocol

There are setter methods such as *setStakingTokenAddress(), setReStakingTokenAddress(), setTransferAddress(), setTreasuryAddress(), setReStakingManagerAddress(),* etc. that are under admin control. The impacts of the admin being compromised maliciously range from protocol malfunction to draining the treasury.

Systemic risks

TRST-SR-1 stKaia solvency of *ReStakingTransfer* contract is dependent on external actions

The stKaia solvency of the *ReStakingTransfer* contract for exchange against gcKlay, stKlay and sKlay is maintained by manual interaction of the protocol owner.

The tokens gcKlay, stKlay and sKlay are manually withdrawn, unstaked and staked again to re-fill the *ReStakingTransfer* with stKaia.

Any failure to periodically follow this routine can lead to stKaia insolvency of *ReStakingTransfer*.