May 9, 2025

ZAN

Security Assessment BlockRock

Professional Service

Table of Contents

1. Overview

1.1. Executive Summary

BlockRock is a project that allows users to subscribe to specific periods using allowed tokens and claim rewards after the period is released. This report has been prepared for BlockRock to discover issues and vulnerabilities in the source code of this project as well as any contract dependencies that were not part of an officially recognized library.

Conducted by Static Analysis, Formal Verification and Manual Review, we have identified 1 High, 3 Low and 8 Informational issues in commit 0755fce.

The project team has resolved the security vulnerabilities described in H-01 and L-04 in commit 90dacb3. As for the issues described in L-02 and L-03, the project team will not make any modifications. Because according to the business design, users can only subscribe using USDT and USDC, so the aforementioned issues do not exist. For details on the resolution of the informational issues, please refer to the Alleviation section.

1.2. Project Summary

Project Name	BlockRock
Platform	Ethereum
Language	Solidity
Codebase	Audit 1: • https://github.com/BlockRock-win/blockrock-contract/tree/0755fcebb94556b46b6805b16a802ef39a57bfb6 Final Audit: • https://github.com/BlockRock-win/blockrock-contract/tree/90dacb38f633b10aab264cb64412c5aa51ad01f0

1.3. Assessment Summary

Delivery Date	May 9, 2025
Audit Methodology	Static Analysis, Formal Verification, Manual Review

1.4. Assessment Scope

ID	File	File Hash
1	/blockrock-contract/src/BlockRockToken.sol	8a34dcc9bb75f77fe611d1cf6683c9ca
2	/blockrock-contract/src/BlockRock.sol	4608b06906b540be5c75ee09508639f8

2. Checklist

2.1. Code Security

Reentrancy	DelegateCall	Integer Overflow
Input Validation	Unchecked this.call	Frozen Money
Arbitrary External Call	Unchecked Owner Transfer	Do-while Continue
Right-To-Left-Override Character	Unauthenticated Storage Access	Risk For Weak Randomness
TxOrigin	Missing Checks for Return Values	Diamond Inheritance
ThisBalance	VarType Deduction	Array Length Manipulation
Uninitialized Variable	Shadow Variable	Divide Before Multiply
Affected by Compiler Bug		

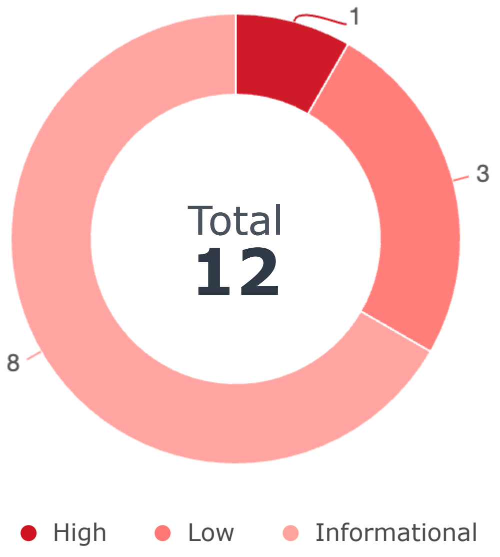
2.2. Optimization Suggestion

Compiler Version	Improper State Variable Modification
Function Visibility	Deprecated Function
Externally Controlled Variables	Code Style
Constant Specific	Event Specific
Return Value Unspecified	Inexistent Error Message
State Variable Defined Without Storage Location	Import Issue
Compare With Timestamp/Block Number/Blockhash	Constructor in Base Contract Not Implemented
Delete Struct Containing the Mapping Type	Usage of '=+'
Paths in the Modifier Not End with "_" or Revert	Non-payable Public Functions Use msg.value
Lack of SafeMath	Compiler Error/Warning
Tautology Issue	Loop Depends on Array Length
Redundant/Duplicated/Dead Code	Code Complexity/Code Inefficiency
Undeclared Resource	Optimizable Return Statement
Unused Resource	

2.3. Business Security

The Code Implementation is Consistent With Comments, Project White Papers and Other Materials
Permission Check
Address Check

3. Findings



Code Security

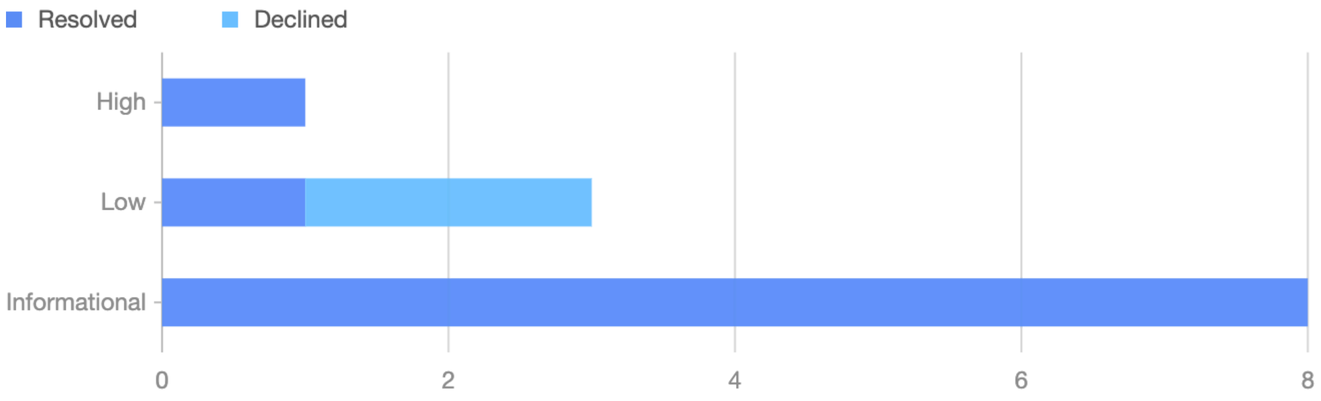
Low: 2

Optimization Suggestion

Informational: 8

Business Security

High: 1, Low: 1



ID	Title	Category	Severity	Status
H-01	The Subscription Does Not Consider Inconsistencies in Token Decimals	Business Security	● High	Resolved
L-02	Different Tokens Receive the Same Reward	Business Security	● Low	Declined
L-03	Contracts Are Vulnerable to Fee-on-Transfer-Token-Related Accounting Issues	Code Security	● Low	Declined
L-04	Use Safer Functions	Code Security	● Low	Resolved
I-05	Insufficient Validity Check	Optimization Suggestion	● Informational	Resolved
I-06	Redundant Validity Check	Optimization Suggestion	● Informational	Resolved
I-07	Ambiguous Error Message	Optimization Suggestion	● Informational	Resolved
I-08	Parameters Should Be Declared as Calldata	Optimization Suggestion	● Informational	Resolved

ID	Title	Category	Severity	Status
I-09	No Check of Address Params with Zero Address	Optimization Suggestion	● Informational	Resolved
I-10	Use ++i/--i Instead of i++/i--	Optimization Suggestion	● Informational	Resolved
I-11	Use != 0 Instead of > 0 for Unsigned Integer Comparison	Optimization Suggestion	● Informational	Resolved
I-12	Using the && Operator Tends to Consume More Gas	Optimization Suggestion	● Informational	Resolved

H-01: The Subscription Does Not Consider Inconsistencies in Token Decimals

High: Business Security

File Location: /blockrock-contract/src/BlockRock.sol:189,191,193

Description

In the `subscribe` function, users subscribe by transferring tokens to the `vault`. When a user transfers an amount of tokens, they receive an equivalent number of shares. However, this does not take into account that different tokens may have different decimals.

For example, in a certain period, both tokenA and tokenB are allowed for subscription. Suppose the decimal for tokenA is 6 and the decimal for tokenB is 8. If a subscriber calls the `subscribe` function and specifies the `amount` parameter as `100,000,000`, if the specified token is tokenA, the actual amount of tokenA transferred to the `vault` by the subscriber is 100 tokens. If the specified token is tokenB, the actual amount of tokenB transferred to the `vault` by the subscriber is 1 token. However, regardless of which token is specified, the subscriber will ultimately receive an additional `100,000,000` shares.

Therefore, subscribers are more inclined to choose tokens with a larger decimal for subscription in order to acquire more shares with fewer tokens.

/blockrock-contract/src/BlockRock.sol

```solidity
177  function subscribe(uint256 periodIndex, uint256 amount, address token)
     external nonReentrant notContract {
178      SubscriptionPeriod storage period = subscriptionPeriods[periodIndex];
179      require(
180          block.timestamp >= period.startTimestamp && block.timestamp <
             period.endTimestamp,
181          "Invalid subscription time"
182      );
183      require(!period.released, "Period already released");
184      require(period.allowedTokens[token], "Invalid purchase token");
185      require(period.remainingShares > 0, "No remaining shares");
186      require(amount >= period.minInvestmentAmount, "Investment amount is
         below the minimum");
187      require(amount <= period.remainingShares, "Insufficient remaining
         shares");
188
189      IERC20(token).safeTransferFrom(msg.sender, period.vault, amount);
190
191      period.remainingShares -= amount;
192      period.totalPurchasedAmount += amount;
193      period.ledger[msg.sender].subscribedShares += amount;
194
195      period.ledger[msg.sender].purchasedTokenAmounts[token] += amount;
196      period.ledger[msg.sender].claimed = false;
197
198      period.totalTokenInvestments[token] += amount;
199      emit UserSubscribed(msg.sender, periodIndex, amount, token);
}
```

Recommendation

It is recommended to handle the token's decimal before updating shares. For example, the number of shares should be calculated as `amount / decimalOfToken`.

Alleviation

Resolved in commit 90dacb3.

L-02: Different Tokens Receive the Same Reward

Low: Business Security

File Location: /blockrock-contract/src/BlockRock.sol:193,195,340

Description

From the `_calculateClaimAmount` function, it can be seen that when calculating the rewards a user can claim, `subscribedShares` is used directly for the computation, which overlooks the impact of different token values on reward distribution.

For example, in a certain period, both tokenA and tokenB are allowed for subscription, and tokenA is valued higher than tokenB. When a subscriber calls the `subscribe` function with the `amount` parameter set to `100`, regardless of whether tokenA or tokenB is specified, they will receive 100 shares. Therefore, when calculating the rewards, the amount they can claim will also be the same.

As a result, subscribers are more inclined to choose the lower-value token for subscriptions in order to acquire more reward tokens with a smaller value token.

/blockrock-contract/src/BlockRock.sol

```
177  function subscribe(uint256 periodIndex, uint256 amount, address token) external nonReentrant notContract {
178      SubscriptionPeriod storage period = subscriptionPeriods[periodIndex];
179      require(
180          block.timestamp >= period.startTimestamp && block.timestamp < period.endTimestamp,
181          "Invalid subscription time"
182      );
183      require(!period.released, "Period already released");
184      require(period.allowedTokens[token], "Invalid purchase token");
185      require(period.remainingShares > 0, "No remaining shares");
186      require(amount >= period.minInvestmentAmount, "Investment amount is below the minimum");
187      require(amount <= period.remainingShares, "Insufficient remaining shares");
188
189      IERC20(token).safeTransferFrom(msg.sender, period.vault, amount);
190
191      period.remainingShares -= amount;
192      period.totalPurchasedAmount += amount;
193      period.ledger[msg.sender].subscribedShares += amount;
194
195      period.ledger[msg.sender].purchasedTokenAmounts[token] += amount;
196      period.ledger[msg.sender].claimed = false;
197
198      period.totalTokenInvestments[token] += amount;
199      emit UserSubscribed(msg.sender, periodIndex, amount, token);
200  }
```

/blockrock-contract/src/BlockRock.sol

```
function _calculateClaimAmount(
    SubscriptionPeriod storage period,
    UserSubscription storage userSub
)
    internal
    view
    returns (uint256)
{
    require(period.released, "Period not released");
    require(period.totalPurchasedAmount > 0, "No investments in this
    period");
    require(period.claimTimestamp <= block.timestamp, "Claim time not
    reached");
    require(!userSub.claimed, "No claimable tokens");
    return (userSub.subscribedShares * period.rewardAmount) / period.
    totalPurchasedAmount;
}
```

Recommendation

It is recommended to clarify whether the above issues meet business expectations. If they do not meet expectations, it is advisable to establish a reward distribution mechanism for tokens of different values.

Alleviation

According to the business design, users can only subscribe using USDT and USDC, both of which are stablecoins. Therefore, the aforementioned issue does not exist. Consequently, the project team will not make any modifications regarding this issue.

L-03: Contracts Are Vulnerable to Fee-on-Transfer-Token-Related Accounting Issues

Low: Code Security

File Location: /blockrock-contract/src/BlockRock.sol:189,191

Description

Contracts interacting with fee-on-transfer tokens may incorrectly assume the amount sent is the amount received by the recipient. This assumption can lead to discrepancies in balance tracking and accounting, especially in decentralized finance (DeFi) applications.

/blockrock-contract/src/BlockRock.sol

```
189         IERC20(token).safeTransferFrom(msg.sender, period.vault, amount);
190
191         period.remainingShares -= amount;
192         period.totalPurchasedAmount += amount;
193         period.ledger[msg.sender].subscribedShares += amount;
```

Recommendation

Implementing robust accounting mechanisms within contracts that interact with FoT tokens is crucial. These mechanisms must account for the transaction fee deducted during the transfer, ensuring accurate balance and transaction records.

Alleviation

According to the business design, users can only subscribe using USDT and USDC, both of which are not FoT tokens. Therefore, the aforementioned issue does not exist. Consequently, the project team will not make any modifications regarding this issue.

L-04: Use Safer Functions

Low: Code Security

File Location: /blockrock-contract/src/BlockRock.sol:230

Description

When calling the `transfer`, `transferFrom`, and `approve` functions in the ERC20 contract, there are some contracts that are not fully implemented in accordance with the ERC20 standard. In order to more comprehensively judge whether the call result meets expectations or to be compatible with different ERC20 contracts, it is recommended to use the `safeTransfer`, `safeTransferFrom`, `safeIncreaseAllowance` / `safeDecreaseAllowance` function to call.

/blockrock-contract/src/BlockRock.sol

```
228         userSub.claimed = true;
229         require(claimAmount > 0, "No tokens to claim");
230         period.rewardToken.transfer(msg.sender, claimAmount);
231         emit Claimed(msg.sender, periodIndex, claimAmount);
232     }
```

Recommendation

It is recommended to use the `safeTransfer`, `safeTransferFrom`, and `safeApprove` functions in SafeERC20 to call the `transfer`, `transferFrom`, and `approve` functions in the ERC20 contract.

Alleviation

Resolved in commit 90dacb3.

I-05: Insufficient Validity Check

Informational: Optimization Suggestion

File Location: /blockrock-contract/src/BlockRock.sol:126,146

Description

The `claimTimestamp` parameter of the `newPeriod` function must be validated to ensure that `claimTimestamp > endTimestamp`, given that the claim operation may only be performed subsequent to the end of the subscription period.

/blockrock-contract/src/BlockRock.sol

```
118     function newPeriod(
119         string memory name,
120         string memory description,
121         string memory logo,
122         address[] memory allowedTokens,
123         uint256 totalShares,
124         uint256 startTimestamp,
125         uint256 endTimestamp,
126         uint256 claimTimestamp,
127         uint256 minInvestmentAmount,
128         address rewardToken,
129         address vault
130     )
131         external
132         onlyOperator
133     {
134         require(startTimestamp > block.timestamp, "Start time must be in
            the future");
135         require(endTimestamp > startTimestamp, "End time must be after
            start time");
136         require(totalShares > 0, "Total shares must be greater than 0");
137         require(minInvestmentAmount > 0, "Minimum investment amount must
            be greater than 0");
138         require(rewardToken != address(0), "Reward token address cannot
            be zero");
139
140         subscriptionPeriods.push();
141         SubscriptionPeriod storage period = subscriptionPeriods
            [subscriptionPeriods.length - 1];
142         period.totalShares = totalShares;
143         period.remainingShares = totalShares;
144         period.startTimestamp = startTimestamp;
145         period.endTimestamp = endTimestamp;
146         period.claimTimestamp = claimTimestamp;
```

Recommendation

Implement the appropriate validation for the `claimTimestamp` parameter.

Alleviation

Resolved in commit 90dacb3.

I-06: Redundant Validity Check

Informational: Optimization Suggestion

File Location:
/blockrock-contract/src/BlockRock.sol:261
/blockrock-contract/src/BlockRockToken.sol:26,35

Description

The validity check at line 261 in `BlockRock.sol` is redundant, as it duplicates the verification process already present within the `Ownable` contract inherited by the `BlockRockToken` contract. Similarly, the check at line 26 in `BlockRockToken.sol` is a duplication of the verification found in the inherited `Ownable` contract. Furthermore, the check at line 35 mirrors the verification implemented within the inherited `ERC20` contract.

/blockrock-contract/src/BlockRock.sol

```
260     function transferTokenOwnership(address rewardToken, address
        newOwner) external onlyOwner {
261         require(newOwner != address(0), "Invalid owner address");
262         BlockRockToken(rewardToken).transferOwnership(newOwner);
263         emit TransferTokenOwnership(newOwner);
264     }
```

/blockrock-contract/src/BlockRockToken.sol

```
18      constructor(
19          string memory name,
20          string memory symbol,
21          address initialOwner
22      )
23          ERC20(name, symbol)
24          Ownable(initialOwner)
25      {
26          require(initialOwner != address(0), "Invalid owner address");
27      }
```

/blockrock-contract/src/BlockRockToken.sol

```
34      function mint(address recipient, uint256 value) external onlyOwner {
35          require(recipient != address(0), "Invalid recipient address");
36          require(value > 0, "Value must be greater than 0");
37          _mint(recipient, value);
38      }
```

Recommendation

Eliminate these redundant `require` statements.

Alleviation

Resolved in commit 90dacb3.

I-07: Ambiguous Error Message

Informational: Optimization Suggestion

File Location: /blockrock-contract/src/BlockRock.sol:90

Description

The `require` statement at line 90 serves to ensure that the present caller is an Externally Owned Account (EOA) rather than a Contract Account (CA), yet the error message is misleading.

/blockrock-contract/src/BlockRock.sol

```
88      modifier notContract() {
89          require(!_isContract(msg.sender), "Contract not allowed");
90          require(msg.sender == tx.origin, "Proxy contract not allowed");
91          _;
92      }
```

Recommendation

Revise the error message to more accurately reflect the actual code implementation.

Alleviation

Resolved in commit 90dacb3.

I-08: Parameters Should Be Declared as Calldata

Informational: Optimization Suggestion

File Location: /blockrock-contract/src/BlockRock.sol:119,120,121,122

Description

When the compiler parses the external or public function, it can directly read the function parameters from calldata. Setting it to other storage locations may waste gas. About 300-400 gas can be saved with optimization turned off while 120-150 gas can be saved vice versa.

/blockrock-contract/src/BlockRock.sol

```
117      */
118     function newPeriod(
119         string memory name,
120         string memory description,
121         string memory logo,
122         address[] memory allowedTokens,
123         uint256 totalShares,
124         uint256 startTimestamp,
```

Recommendation

In external or public functions, the storage location of function parameters should be set to calldata to save gas.

Alleviation

Resolved in commit 90dacb3.

I-09: No Check of Address Params with Zero Address

Informational: Optimization Suggestion

File Location: /blockrock-contract/src/BlockRock.sol:100,122,129,238

Description

The input parameters `_operator`, `_admin`, `vault`, `allowedTokens[i]` and `_operator` which are of the address type in these functions do not use the zero address for verification.

/blockrock-contract/src/BlockRock.sol

```
98       * @param _admin The address of the admin.
99       */
100     constructor(address contractOwner, address _operator, address
        _admin) Ownable(contractOwner) {
101         operator = _operator;
102         admin = _admin;
```

/blockrock-contract/src/BlockRock.sol

```
116      * @param vault The vault address for the subscription period.
117      */
118     function newPeriod(
119         string memory name,
120         string memory description,
121         string memory logo,
122         address[] memory allowedTokens,
123         uint256 totalShares,
124         uint256 startTimestamp,
125         uint256 endTimestamp,
126         uint256 claimTimestamp,
127         uint256 minInvestmentAmount,
128         address rewardToken,
129         address vault
130     )
```

/blockrock-contract/src/BlockRock.sol

```
236      * @param _operator The address of the new operator.
237      */
238     function setOperator(address _operator) external onlyOwner {
239         address oldOperator = operator;
240         operator = _operator;
```

Recommendation

It is recommended to perform zero address verification on the input parameters of the address type.

Alleviation

Resolved in commit 90dacb3.

I-10: Use ++i/--i Instead of i++/i--

Informational: Optimization Suggestion

File Location: /blockrock-contract/src/BlockRock.sol:152

Description

Compared with `i++`, `++i` can save about 5 gas per use. Compared with `i--`, `--i` can save about 3 gas per use in for loop.

/blockrock-contract/src/BlockRock.sol

```
        period.rewardToken = BlockRockToken(rewardToken);
        period.vault = vault;
        for (uint256 i = 0; i < allowedTokens.length; i++) {
            period.allowedTokens[allowedTokens[i]] = true;
        }
```

Recommendation

It is recommended to use `++i`/`--i` instead of `i++`/`i--` in for loop.

Alleviation

Resolved in commit 90dacb3.

I-11: Use != 0 Instead of > 0 for Unsigned Integer Comparison

Informational: Optimization Suggestion

File Location:
/blockrock-contract/src/BlockRock.sol:136,137,211,352
/blockrock-contract/src/BlockRockToken.sol:36

Description

For unsigned integers, use `!=0` for comparison, which consumes less gas than `>0`. When compiler optimization is turned off, about 3 gas can be saved. When compiler optimization is turned on, no gas can be saved.

/blockrock-contract/src/BlockRock.sol

```
134         require(startTimestamp > block.timestamp, "Start time must be in the future");
135         require(endTimestamp > startTimestamp, "End time must be after start time");
136         require(totalShares > 0, "Total shares must be greater than 0");
137         require(minInvestmentAmount > 0, "Minimum investment amount must be greater than 0");
138         require(rewardToken != address(0), "Reward token address cannot be zero");
```

/blockrock-contract/src/BlockRock.sol

```
135         require(endTimestamp > startTimestamp, "End time must be after start time");
136         require(totalShares > 0, "Total shares must be greater than 0");
137         require(minInvestmentAmount > 0, "Minimum investment amount must be greater than 0");
138         require(rewardToken != address(0), "Reward token address cannot be zero");
139
```

/blockrock-contract/src/BlockRock.sol

```
209         require(block.timestamp >= period.endTimestamp, "Period not ended");
210         require(!period.released, "Period already released");
211         require(mintAmount > 0, "Mint amount must be greater than 0");
212         require(period.totalPurchasedAmount > 0, "No investments in this period");
213
```

/blockrock-contract/src/BlockRock.sol

```
            size := extcodesize(account)
        }
        return size > 0;
    }
}
```

/blockrock-contract/src/BlockRockToken.sol

```
    function mint(address recipient, uint256 value) external onlyOwner {
        require(recipient != address(0), "Invalid recipient address");
        require(value > 0, "Value must be greater than 0");
        _mint(recipient, value);
    }
```

Recommendation

For unsigned integers, it is recommended to use `!=0` instead of `>0` for comparison.

Alleviation

Resolved in commit 90dacb3.

I-12: Using the && Operator Tends to Consume More Gas

Informational: Optimization Suggestion

File Location: /blockrock-contract/src/BlockRock.sol:179

Description

Usage of double `require` will save you around 10 gas with the optimizer enabled.

/blockrock-contract/src/BlockRock.sol

```
177     function subscribe(uint256 periodIndex, uint256 amount, address token) external nonReentrant notContract {
178         SubscriptionPeriod storage period = subscriptionPeriods[periodIndex];
179         require(
180             block.timestamp >= period.startTimestamp && block.timestamp < period.endTimestamp,
181             "Invalid subscription time"
```

Recommendation

Using double `require` instead of operator `&&`.

Alleviation

Resolved in commit 90dacb3.

4. Disclaimer

No description, statement, recommendation or conclusion in this report shall be construed as endorsement, affirmation or confirmation of the project. The security assessment is limited to the scope of work as stipulated in the Statement of Work.

This report is prepared in response to source code, and based on the attacks and vulnerabilities in the source code that already existed or occurred before the date of this report, excluding any new attacks or vulnerabilities that exist or occur after the date of this report. The security assessment are solely based on the documents and materials provided by the customer, and the customer represents and warrants documents and materials are true, accurate and complete.

CONSULTANT DOES NOT MAKE AND HEREBY DISCLAIMS ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, REGARDING THE SERVICES, DELIVERABLES, OR ANY OTHER MATTER PERTAINING TO THIS REPORT.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR AND HEREBY DISCLAIMS MERCHANTABILITY, FITNESS FOR PURPOSE, TITLE, NON-INFRINGEMENT OR NON-APPROPRIATION OF INTELLECTUAL PROPERTY RIGHTS OF A THIRD PARTY, SATISFACTORY QUALITY, ACCURACY, QUALITY, COMPLETENESS, TIMELINESS, RESPONSIVENESS, OR PRODUCTIVITY OF THE SERVICES OR DELIVERABLES.

CONSULTANT EXCLUDES ANY WARRANTY THAT THE SERVICES AND DELIVERABLES WILL BE UNINTERRUPTED, ERROR FREE, FREE OF SECURITY DEFECTS OR HARMFUL COMPONENTS, REVEAL ALL SECURITY VULNERABILITIES, OR THAT ANY DATA WILL NOT BE LOST OR CORRUPTED.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR (A) ANY REPRESENTATIONS MADE BY ANY PERSON REGARDING THE SUFFICIENCY OR SUITABILITY OF SERVICES AND DELIVERABLES IN ANY ACTUAL APPLICATION, OR (B) WHETHER ANY SUCH USE WOULD VIOLATE OR INFRINGE THE APPLICABLE LAWS, OR (C) REVIEWING THE CUSTOMER MATERIALS FOR ACCURACY.

5. Appendix

5.1 Visibility

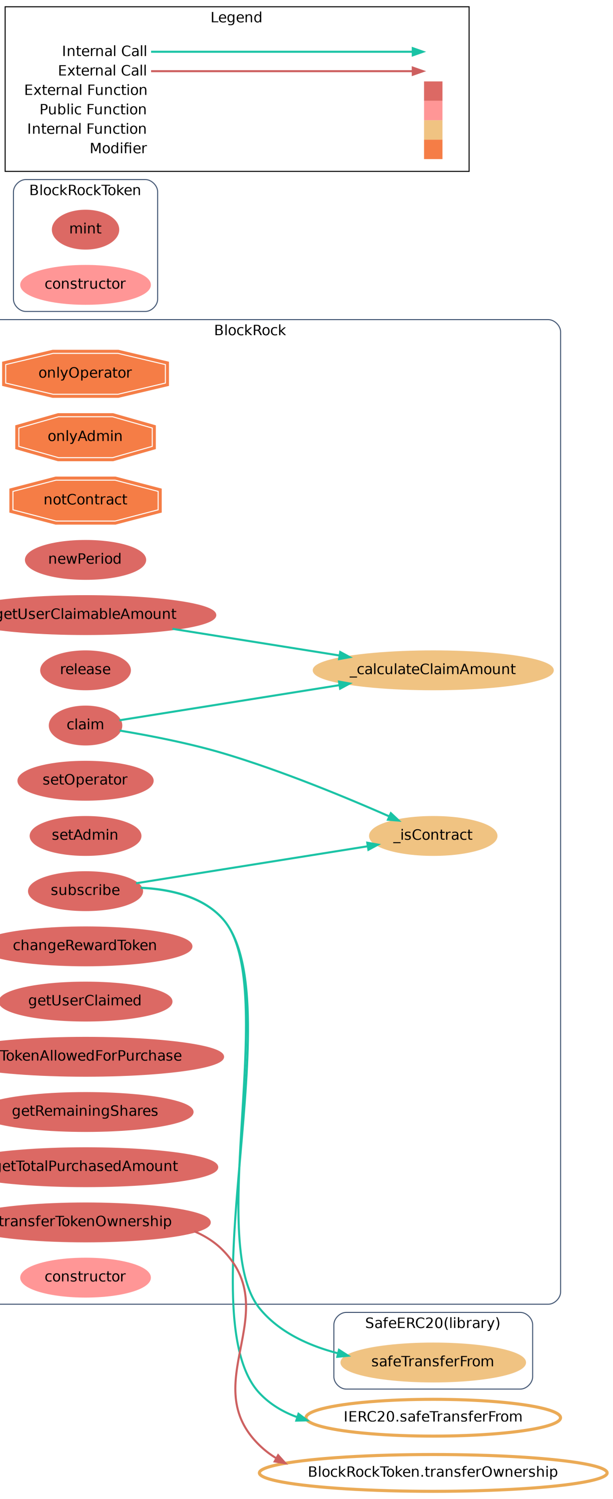
Contract	FuncName	Visibility	Mutability	Modifiers
BlockRockToken	_CTOR_	public	N	
BlockRockToken	mint	external	Y	onlyOwner
BlockRock	_CTOR_	public	Y	
BlockRock	newPeriod	external	Y	onlyOperator
BlockRock	subscribe	external	Y	nonReentrant, notContract
BlockRock	release	external	Y	onlyAdmin, nonReentrant
BlockRock	claim	external	Y	notContract
BlockRock	setOperator	external	Y	onlyOwner
BlockRock	setAdmin	external	Y	onlyOwner
BlockRock	transferTokenOwnership	external	N	onlyOwner
BlockRock	changeRewardToken	external	Y	onlyOwner
BlockRock	getUserClaimed	external	N	
BlockRock	isTokenAllowedForPurchase	external	N	
BlockRock	getRemainingShares	external	N	
BlockRock	getTotalPurchasedAmount	external	N	
BlockRock	getUserClaimableAmount	external	N	
BlockRock	_calculateClaimAmount	internal	N	

Contract	FuncName	Visibility	Mutability	Modifiers
BlockRock	_isContract	internal	N	

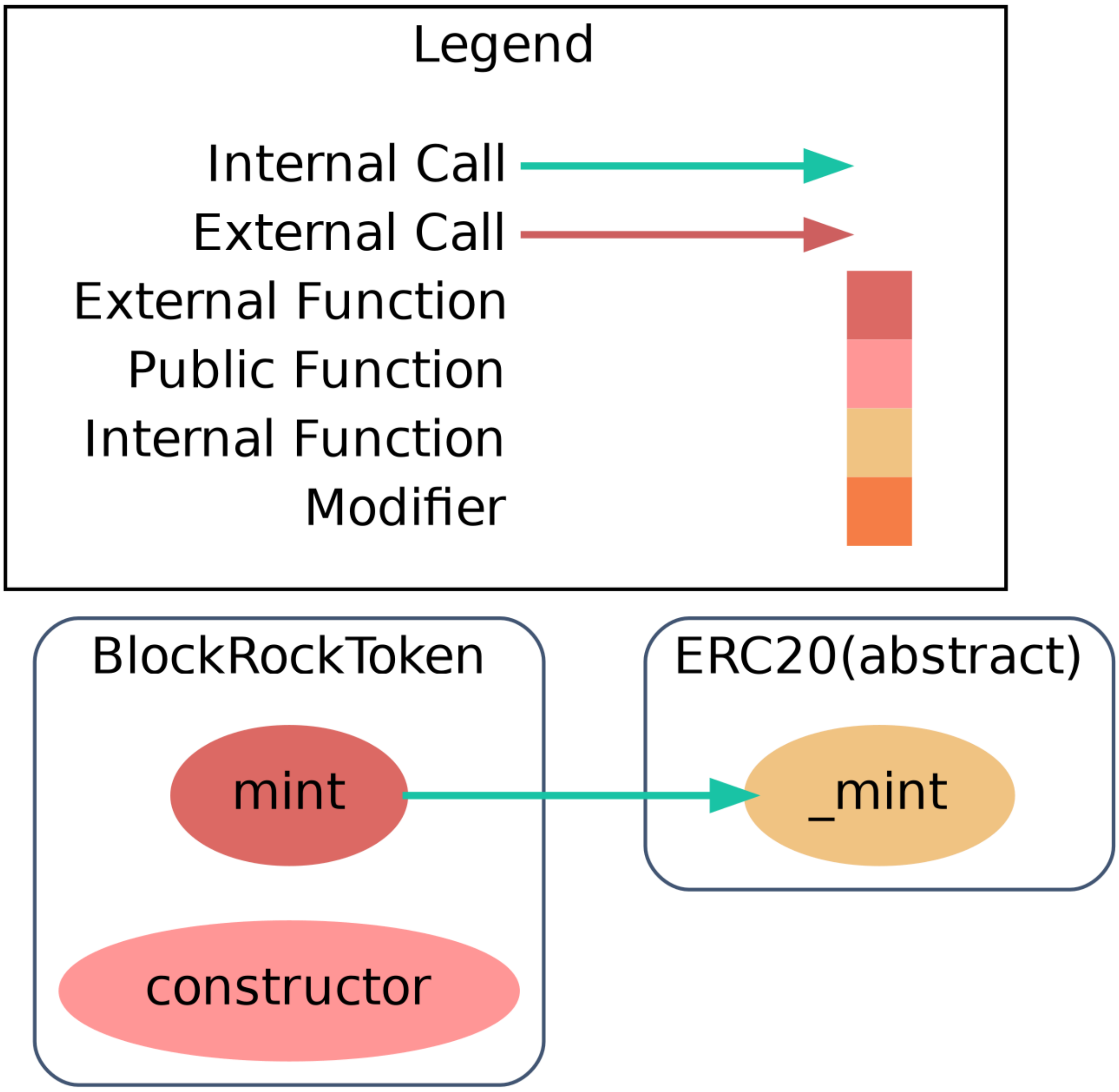
5. Appendix

5.2 Call Graph

BlockRock



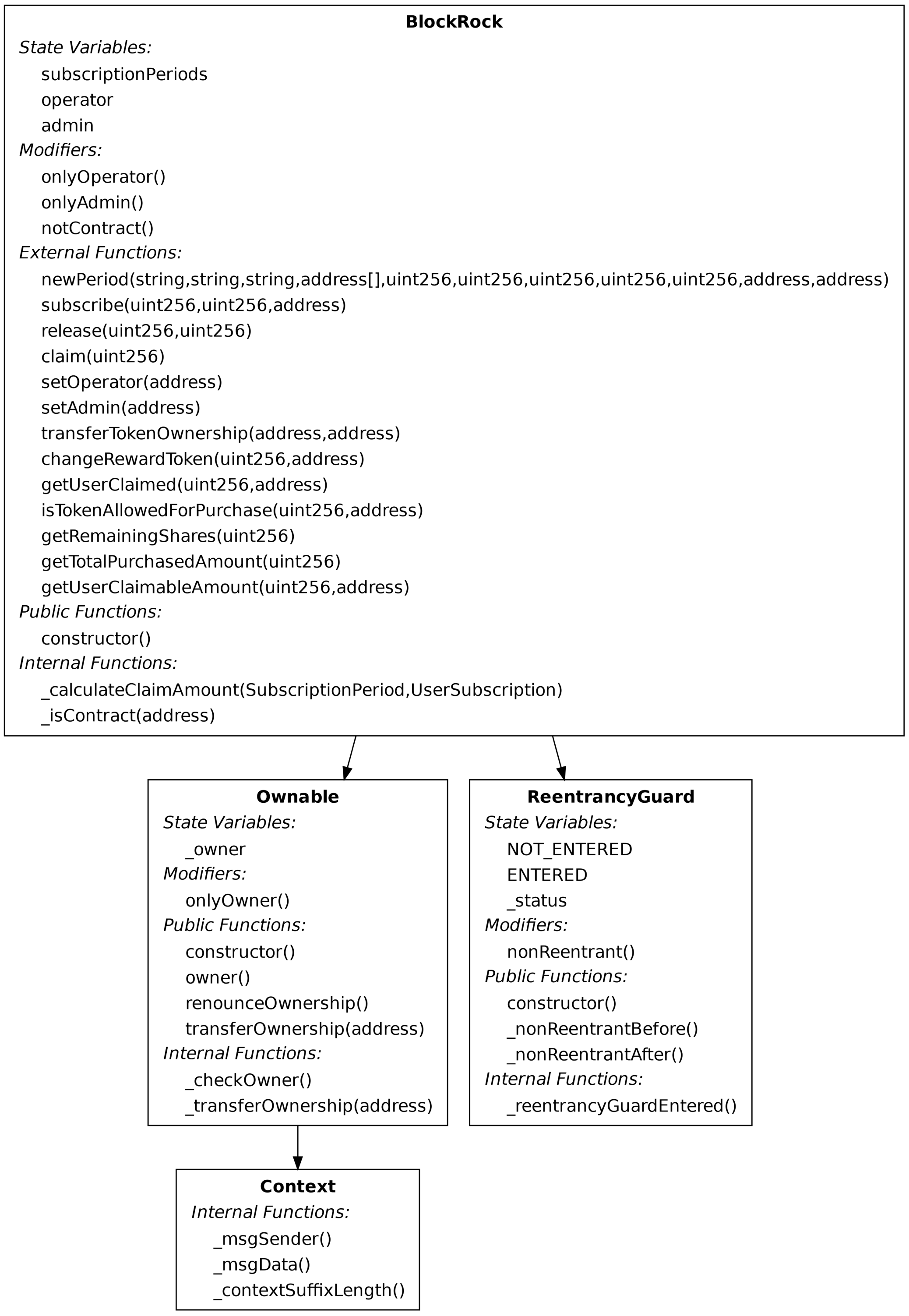
BlockRockToken



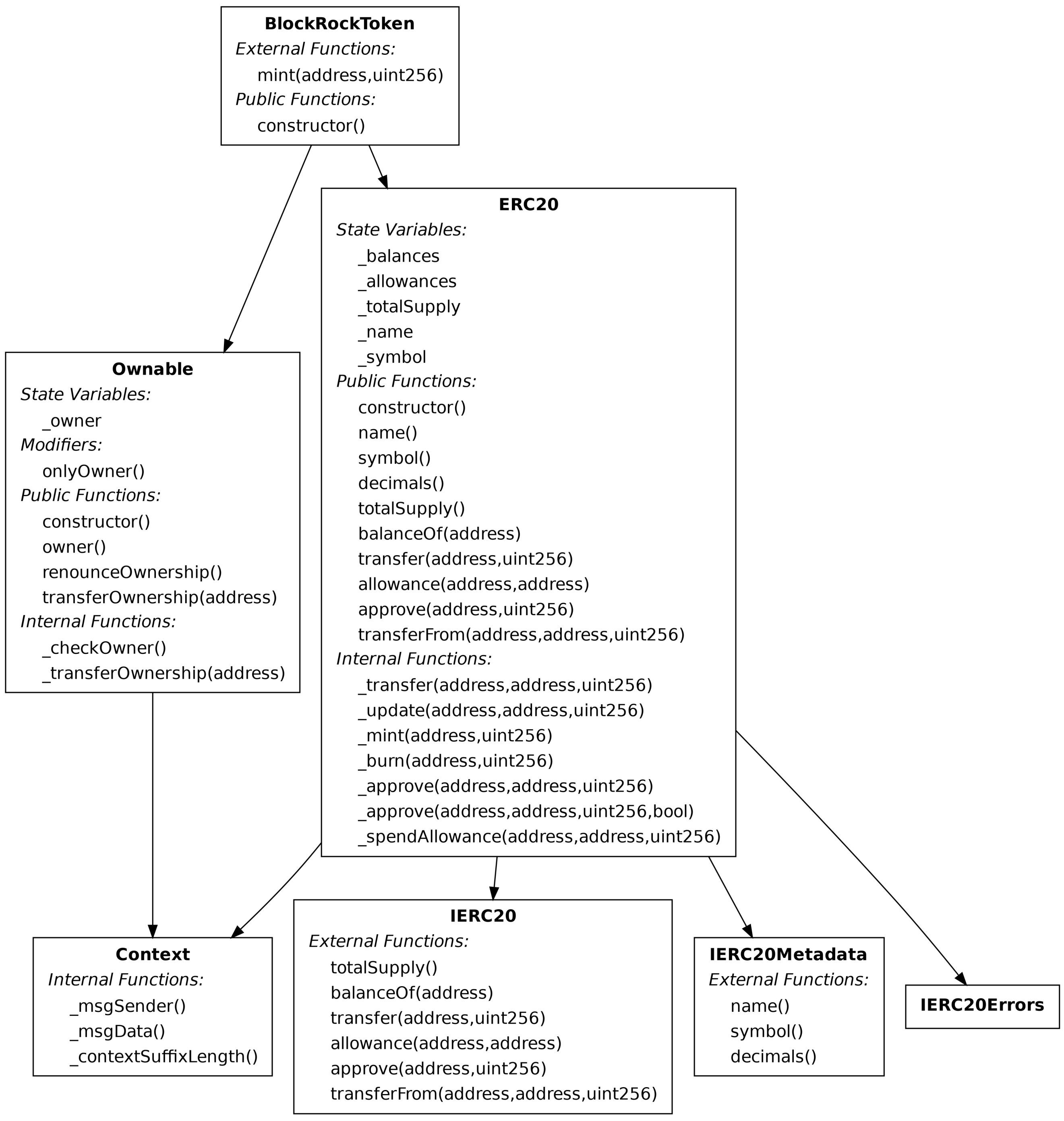
5. Appendix

5.3 Inheritance Graph

BlockRock



BlockRockToken



5.4 Formal Verification Metadata

1. Should the function `subscribe` execute successfully, it is imperative that the subscription period's start timestamp be less than or equal to the current block timestamp and its end timestamp be strictly greater than the current block timestamp.

```
/// #if_succeeds {:msg "Should the function `subscribe` execute
successfully, it is imperative that the subscription period's start
timestamp be less than or equal to the current block timestamp and its end
timestamp be strictly greater than the current block timestamp."}
old(block.timestamp >= subscriptionPeriods[periodIndex].startTimestamp &&
block.timestamp < subscriptionPeriods[periodIndex].endTimestamp);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

2. Upon the successful execution of `subscribe`, the subscription period must not have been previously released.

```
/// #if_succeeds {:msg "Upon the successful execution of `subscribe`, the
subscription period must not have been previously released."}
old(!subscriptionPeriods[periodIndex].released);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

3. Provided the function `subscribe` completes without error, the provided token address must be among those sanctioned for acquisition within this subscription period.

```
/// #if_succeeds {:msg "Provided the function `subscribe` completes without
error, the provided token address must be among those sanctioned for
acquisition within this subscription period."}
old(subscriptionPeriods[periodIndex].allowedTokens[token]);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

4. In the event of a successful execution of `subscribe`, it is essential that the subscription period retains residual shares available for subscription.

```
/// #if_succeeds {:msg "In the event of a successful execution of
`subscribe`, it is essential that the subscription period retains residual
shares available for subscription."}
```

```
old(subscriptionPeriods[periodIndex].remainingShares > 0);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

5. In the case where the `subscribe` function executes successfully, the investment amount utilized must meet or exceed the minimum investment requirement stipulated by the subscription period.

```
/// #if_succeeds {:msg "In the case where the `subscribe` function executes
successfully, the investment amount utilized must meet or exceed the
minimum investment requirement stipulated by the subscription period."}
old(amount >= subscriptionPeriods[periodIndex].minInvestmentAmount);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

6. If the `subscribe` operation concludes successfully, the requested subscription amount should not surpass the quantity of remaining shares in the subscription period.

```
/// #if_succeeds {:msg "If the `subscribe` operation concludes
successfully, the requested subscription amount should not surpass the
quantity of remaining shares in the subscription period."} old(amount <=
subscriptionPeriods[periodIndex].remainingShares);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

7. Should the `subscribe` function conclude with success, the total number of remaining shares in the subscription period ought to diminish precisely by the amount subscribed.

```
/// #if_succeeds {:msg "Should the `subscribe` function conclude with
success, the total number of remaining shares in the subscription period
ought to diminish precisely by the amount subscribed."}
subscriptionPeriods[periodIndex].remainingShares ==
old(subscriptionPeriods[periodIndex].remainingShares - amount);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

8. Upon the successful conclusion of `subscribe`, the cumulative total of purchased tokens for this subscription period should increase exactly by the amount

transferred.

```
/// #if_succeeds {:msg "Upon the successful conclusion of `subscribe`, the
cumulative total of purchased tokens for this subscription period should
increase exactly by the amount transferred."}
subscriptionPeriods[periodIndex].totalPurchasedAmount ==
old(subscriptionPeriods[periodIndex].totalPurchasedAmount + amount);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

9. Provided the `subscribe` function runs to completion without failure, the user's subscribed shares should reflect an increment equivalent to the amount transferred.

```
/// #if_succeeds {:msg "Provided the `subscribe` function runs to
completion without failure, the user's subscribed shares should reflect an
increment equivalent to the amount transferred."}
subscriptionPeriods[periodIndex].ledger[msg.sender].subscribedShares ==
old(subscriptionPeriods[periodIndex].ledger[msg.sender].subscribedShares +
amount);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

10. After a successful execution of `subscribe`, the user's individual record for the specific token used in subscription should reflect an updated increase corresponding to the amount transferred.

```
/// #if_succeeds {:msg "After a successful execution of `subscribe`, the
user's individual record for the specific token used in subscription should
reflect an updated increase corresponding to the amount transferred."}
subscriptionPeriods[periodIndex].ledger[msg.sender].purchasedTokenAmounts[toke
==
old(subscriptionPeriods[periodIndex].ledger[msg.sender].purchasedTokenAmounts[
+ amount);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

11. Following a successful execution of `subscribe`, the total investment attributed to the specific token in this subscription period should have increased by the exact amount transferred.

```
/// #if_succeeds {:msg "Following a successful execution of `subscribe`,
the total investment attributed to the specific token in this subscription
period should have increased by the exact amount transferred."}
subscriptionPeriods[periodIndex].totalTokenInvestments[token] ==
old(subscriptionPeriods[periodIndex].totalTokenInvestments[token] + amount);
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

12. Upon the successful execution of `subscribe`, the 'claimed' status for the user in this subscription period must remain false or be set to false.

```
/// #if_succeeds {:msg "Upon the successful execution of `subscribe`, the
'claimed' status for the user in this subscription period must remain false
or be set to false."}
!subscriptionPeriods[periodIndex].ledger[msg.sender].claimed;
function subscribe(uint256 periodIndex, uint256 amount, address token)
external nonReentrant notContract {
```

Passed.

13. If the `claim` function executes successfully, the subscription period must have been released prior to execution.

```
/// #if_succeeds {:msg "If the `claim` function executes successfully, the
subscription period must have been released prior to execution."}
old(subscriptionPeriods[periodIndex].released);
function claim(uint256 periodIndex) external notContract {
```

Passed.

14. Upon successful execution of `claim`, the total purchased amount in the subscription period must be greater than zero.

```
/// #if_succeeds {:msg "Upon successful execution of `claim`, the total
purchased amount in the subscription period must be greater than zero."}
old(subscriptionPeriods[periodIndex].totalPurchasedAmount > 0);
function claim(uint256 periodIndex) external notContract {
```

Passed.

15. Should the `claim` function execute without failure, the current block timestamp must be greater than or equal to the claim timestamp of the subscription period.

```
/// #if_succeeds {:msg "Should the `claim` function execute without
failure, the current block timestamp must be greater than or equal to the
claim timestamp of the subscription period."} old(block.timestamp >=
subscriptionPeriods[periodIndex].claimTimestamp);
function claim(uint256 periodIndex) external notContract {
```

Passed.

16. After a successful execution of `claim`, the user's claimed status for this subscription period must be set to true.

```
/// #if_succeeds {:msg "After a successful execution of `claim`, the user's
claimed status for this subscription period must be set to true."}
subscriptionPeriods[periodIndex].ledger[msg.sender].claimed;
function claim(uint256 periodIndex) external notContract {
```

Passed.

17. Provided the `claim` operation completes successfully, the calculated claim amount must be greater than zero.

```
/// #if_succeeds {:msg "Provided the `claim` operation completes
successfully, the calculated claim amount must be greater than zero."}
$result > 0;
function claim(uint256 periodIndex) external notContract {
```

Passed.

18. In the case where `claim` executes successfully, the transferred reward tokens should match the calculated claimable amount for the user.

```
/// #if_succeeds {:msg "In the case where `claim` executes successfully,
the transferred reward tokens should match the calculated claimable amount
for the user."}
subscriptionPeriods[periodIndex].rewardToken.balanceOf(msg.sender) ==
old(subscriptionPeriods[periodIndex].rewardToken.balanceOf(msg.sender)) +
$result;
function claim(uint256 periodIndex) external notContract {
```

Passed.