Smart Contract
Security Audit Report

Prepared for Binance

Prepared by Supremacy

November 13, 2024

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the Wrapped Beacon ETH, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Client

Binance is the world's largest crypto exchange by trading volume, with $76 billion daily trading volume on Binance exchange as of August 2022, and 90 million customers worldwide. The platform has established itself as a trusted member of the crypto space, where users can buy, sell and store their digital assets, as well as access over 350 cryptocurrencies listed and thousands of trading pairs. The Binance ecosystem now comprises of Binance Exchange, Labs, Launchpad, Info, Academy, Research, Trust Wallet, Charity, NFT and more.

Wrapped Beacon ETH (WBETH) represents your staked Ethereum (ETH) and the staking reward received, in a tradable and transferable form. WBETH accumulates ETH staking rewards by growing in value in relation to ETH, even when it is used in Binance products or DeFi Projects.

Item	Description
Client	Binance
Website	https://www.binance.com
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

- Repository: https://github.com/earn-tech-git/wbeth/tree/develop_unwrap/contracts/wrapped-tokens
- Commit Hash: 279917103288e378765d50993165e8805d7e639e

And this is the commit hash after all fixes for the issues found in the security audit have been checked in:

- Repository: https://github.com/earn-tech-git/wbeth/tree/EARN-17197/1/contracts/wrapped-tokens
- Commit Hash: 80a63f56b8fd95df623504fd4a8548eec51853f5

1.3 Changelogs

Version	Date	Description
0.1	August 29, 2023	Initial Draft
0.2	August 30, 2023	Release Candidate #1
1.0	September 01, 2023	Final Release
1.1	November 13, 2024	Post-Final Release #1

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology precipitation and innovative research.

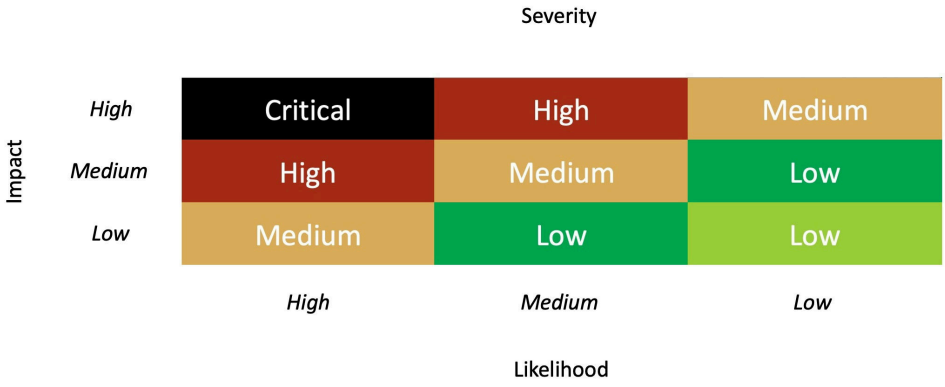
We are reachable at X (https://x.com/SupremacyHQ), or Email (contact@supremacy.email).

1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.



As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Medium	Improperly hard-coded	Fixed
2	Medium	Centralized risk	Confirmed
3	Medium	The potential bypass risk with the AML	Undetermined
4	Medium	Rescueable’s centralization risk	Fixed
5	Low	The potential bypass risk of Access control	Fixed
6	Low	The potential freezing of funds	Acknowledged
7	Informational	Lack of isContract validation	Confirmed
8	Informational	Lack of original address validation	Confirmed
9	Informational	Gas optimization	Acknowledged
10	Informational	Claim flag optimization	Acknowledged
11	Informational	Best Practice	Fixed
12	Informational	Defined local variables not well utilized	Fixed
13	Informational	Code optimization	Acknowledged

2.1 Medium

1. Improperly hard-coded [Medium]

`Severity: Medium` `Likelihood: Low` `Impact: High`

`Status: Fixed`

Description

The constants `_ETH_ADDRESS` and `_UNWRAP_ETH_ADDRESS` have been set to Testnet addresses in the `WrapTokenV2BSC` and `WrapTokenV2ETH` smart contract(s).

```
7  contract WrapTokenV2BSC is StakedTokenV2 {
8      /**
9       * @dev ETH contract address on current chain.
10      */
11     address public constant _ETH_ADDRESS =
   0xE7bCB9e341D546b66a46298f4893f5650a56e99E;
12
13     /**
14     * @dev UNWRAP ETH contract address on current chain.
15      */
16     address public constant _UNWRAP_ETH_ADDRESS =
   0x5159fC6E2487828904eB1254B46365315063c86D;
```

WrapTokenV2BSC.sol

```
7  contract WrapTokenV2ETH is StakedTokenV2 {
8      /**
9       * @dev gas limit of eth transfer.
10      */
11     uint256 private constant _ETH_TRANSFER_GAS = 5000;
12
13     /**
14     * @dev UNWRAP ETH contract address on current chain.
15      */
16     address public constant _UNWRAP_ETH_ADDRESS =
   0x6092ff3683AA223464F97e93feF716dCdB146de3;
```

WrapTokenV2ETH.sol

Recommendation: Consider configuring the correct addresses for the unwrap or changing the variable type and adding functions before deploying to the Mainnet.

2. Centralized risk [Medium]

`Severity: Medium` `Likelihood: Low` `Impact: High`

`Status: Confirmed`

Description

wBETH is an interest-bearing asset, which, according to its business logic, should only be created when a user pledges ETH via the Deposit function, and in the `StakedTokenV2::mint()` and `FiatTokenV1::mint()` privilege functions. Privileged accounts can directly mint wBETH, obviously with some degree of centralization risk.

```
110     /**
111      * @dev Function to mint tokens to msg.sender
112      * @param amount to mint
113      */
114     function mint(uint256 amount)
115         external
116         onlyMinters
117         returns (bool)
118     {
119         uint256 mintingAllowedAmount = minterAllowed[msg.sender];
120         require(
121             amount <= mintingAllowedAmount,
122             "StakedTokenV1: mint amount exceeds minterAllowance"
123         );
124 
125         _mint(msg.sender, amount);
126 
127         minterAllowed[msg.sender] = mintingAllowedAmount.sub(amount);
128         return true;
129     }
```

StakedTokenV2.sol

Recommendation: Remove such privileged functions.

3. The potential bypass risk with the AML [Medium]

`Severity: Medium` `Likelihood: Low` `Impact: High`

`Status: Undetermined`

Description

wBETH is a compliant product, which means it will abide by compliance and AML rules. The Anti-Money Laundering mechanism is the core of the compliance system. As far as `WrapToken` and `UnwrapToken` are concerned, since they are two modules in the same business, their respective blacklist (Anti-Money Laundering) mechanisms will also be separated and not in the smart Maintaining the system uniformly at the contract level may have unintended consequences.

For example:

I. Omitting a freeze on UnwrapToken where the regulated person had previously sensed the impending restriction by monitoring Mempool and bypassed the AML restriction by initiating the Unwrap request in advance or by exchanging the asset directly in the liquidity pool.

II. Omitting the freeze on WrapToken, and the regulated person can bypass the AML restriction by exchanging assets directly in the liquidity pool.

Recommendation:

The temporary solution is to unify the request for **freezing (blacklisting)** transactions under the chain, but in the long run, the optimal solution is to construct an AML system contract, and the two smart contracts, `WrapToken` and `UnwrapToken`, will obtain the AML restriction through the form of external calls.

4. Rescueable's centralization risk [Medium]

`Severity: Medium` `Likelihood: Low` `Impact: High`

`Status: Fixed`

Description

It is important to note that this vulnerability was in code that was out of scope for this audit and would have likely gone unnoticed if not for the excellent work of the our research team.

At the contract level on the Ethereum platform, any asset transfer involving ETH, either to or from, is bound to involve calls to `.transfer()`, `.send()`, `.call()`.

In the BNB Chain platform, it uses the `safe` class function to transfer ETH assets based on ERC20.

We found a function in the `WrapTokenV2BSC` contract of the BNB Chain platform that can transfer out ETH assets deposited by all users.

```
54      /**
55       * @notice Rescue ERC20 tokens locked up in this contract.
56       * @param tokenContract ERC20 token contract address
57       * @param to        Recipient address
58       * @param amount    Amount to withdraw
59       */
60      function rescueERC20(
61          IERC20 tokenContract,
62          address to,
63          uint256 amount
64      ) external onlyRescuer {
65          tokenContract.safeTransfer(to, amount);
66      }
```

Rescuable.sol

Recommendation: Add the code.

```
54      /**
55       * @notice Rescue ERC20 tokens locked up in this contract.
56       * @param tokenContract ERC20 token contract address
57       * @param to        Recipient address
58       * @param amount    Amount to withdraw
59       */
60      function rescueERC20(
61          IERC20 tokenContract,
62          address to,
63          uint256 amount
64      ) external onlyRescuer {
65  +       require(address(tokenContract) != _ETH_ADDRESS);
66          tokenContract.safeTransfer(to, amount);
67      }
```

Rescuable.sol

2.2 Low

5. The potential bypass risk of Access control [Low]

`Severity: Low` `Likelihood: Low` `Impact: Medium`

`Status: Fixed`

Description

`UnwrapTokenV1::claimWithdraw()` receive a `uint256` type parameter named `_index` to index to `_allocateIndex` via `_userRequests[_index]` and fetch the withdrawal request according to this global index. However, access control is not well implemented in this function, which means that some extreme values, such as `UnwrapTokenV1::claimWithdraw(0)`, with an empty array of `_userRequests[]`, can cause access to `_allocateIndex = 0`. But here, due to the presence of `_userRequests.pop()`, if the user's `_userRequests[]` is an empty array, it will result in a revert.

```
255     /**
256      * @dev claim the allocated eth
257      * @param _index the index to claim
258      * @return the eth amount
259      */
260     function claimWithdraw(uint256 _index) external whenNotPaused
261             notBlacklisted(msg.sender) returns (uint256)
262     {
263         address user = msg.sender;
264         uint256[] storage _userRequests = userWithdrawRequests[user];
265         require(_index < _userRequests.length, "Invalid index");
266 
267         uint256 _allocateIndex = _userRequests[_index];
268         WithdrawRequest storage _withdrawRequest = withdrawRequests[_allocateIndex];
269         uint256 _ethAmount = _withdrawRequest.ethAmount;
270 
271         require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime), "Claim time not reach");
272         require(_withdrawRequest.allocated, "Not allocated yet");
273         require(_withdrawRequest.claimTime == 0, "Already claim yet");
274         require(_getCurrentBalance() >= _ethAmount, "Not enough balance");
275 
276         if (_userRequests.length > 1) {
277             _userRequests[_index] = _userRequests[_userRequests.length - 1];
278         }
279         _userRequests.pop();
280 
281         _withdrawRequest.claimTime = block.timestamp;
282         _transferEth(msg.sender, _ethAmount);
283         emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
284         return _ethAmount;
285     }
```

UnwrapTokenV1.sol

Nevertheless, this is still not an effective access control policy.

Recommendation: Adding Access Controls.

```
255      /**
256       * @dev claim the allocated eth
257       * @param _index the index to claim
258       * @return the eth amount
259       */
260      function claimWithdraw(uint256 _index) external whenNotPaused
261              notBlacklisted(msg.sender) returns (uint256)
262      {
263          address user = msg.sender;
264          uint256[] storage _userRequests = userWithdrawRequests[user];
265          require(_index < _userRequests.length, "Invalid index");
266
267          uint256 _allocateIndex = _userRequests[_index];
268          WithdrawRequest storage _withdrawRequest =
    withdrawRequests[_allocateIndex];
269          uint256 _ethAmount = _withdrawRequest.ethAmount;
270
271  +       require(_withdrawRequest.recipient == user, "Wrong recipient");
272          require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime),
    "Claim time not reach");
273          require(_withdrawRequest.allocated, "Not allocated yet");
274          require(_withdrawRequest.claimTime == 0, "Already claim yet");
275          require(_getCurrentBalance() >= _ethAmount, "Not enough balance");
276
277          if (_userRequests.length > 1) {
278              _userRequests[_index] = _userRequests[_userRequests.length - 1];
279          }
280          _userRequests.pop();
281
282          _withdrawRequest.claimTime = block.timestamp;
283          _transferEth(msg.sender, _ethAmount);
284          emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
285          return _ethAmount;
286      }
```

UnwrapTokenV1.sol

6. The potential freezing of funds [Low]

`Severity: Low` `Likelihood: Low` `Impact: Medium`

`Status: Acknowledged`

Description

`Deposit()` is used to pledge Native ETH to mint `wBETH`, while `RequestWithdrawEth()` is the exit mechanism for `wBETH`. The conversion between them is done through a formula, as the exchange rate adjustment is affected by centralization, and if the `exchangeRate()` return value used by the user when they need to withdraw is maliciously controlled, ideally the user's ETH funds will be affected and not dare to withdraw them easily.

```
18      /**
19       * @dev Function to deposit eth to the contract for wBETH
20       * @param referral The referral address
21       */
22      function deposit(address referral) external payable {
23          require(msg.value > 0, "zero ETH amount");
24
```

```
25          // msg.value and exchangeRate are all scaled by 1e18
26          uint256 wBETHAmount =
    msg.value.mul(_EXCHANGE_RATE_UNIT).div(exchangeRate());
27
28          _mint(msg.sender, wBETHAmount);
29
30          emit DepositEth(msg.sender, msg.value, wBETHAmount, referral);
31      }
```

WrapTokenV2ETH.sol

The minting calculation formula is as follows:

$$\_EXCHANGE\_RATE\_UNIT = 1e18$$

$$\text{exchangeRate}() = 1e18$$

$$\text{msg.value} * \_EXCHANGE\_RATE\_UNIT \ / \ \text{exchangeRate}()$$

```
62      /**
63       * @dev Function to withdraw wBETH for eth
64       * @param wbethAmount The wBETH amount
65       */
66      function requestWithdrawEth(uint256 wbethAmount) external {
67          require(wbethAmount > 0, "zero wBETH amount");
68
69          // msg.value and exchangeRate are all scaled by 1e18
70          uint256 ethAmount =
    wbethAmount.mul(exchangeRate()).div(_EXCHANGE_RATE_UNIT);
71          _burn(wbethAmount);
72          IUnwrapTokenV1(_UNWRAP_ETH_ADDRESS).requestWithdraw(msg.sender,
    wbethAmount, ethAmount);
73          emit RequestWithdrawEth(msg.sender, wbethAmount, ethAmount);
74      }
```

WrapTokenV2ETH.sol

The withdrawal calculation formula is as follows:

$$\text{exchangeRate}() = 1$$

$$\_EXCHANGE\_RATE\_UNIT = 1e18$$

$$\text{wbethAmount} * \text{exchangeRate}() \ / \ \_EXCHANGE\_RATE\_UNIT$$

Recommendation: Strict control of exchange rate adjustments.

2.3 Informational

7. Lack of isContract validation [Informational]

`Status: Confirmed`

Description

In the `MintForwarder::initialize()` and `ExchangeRateUpdater::initialize()` functions do not perform strict contract validation on the incoming `newTokenContract` parameter, which may result in unintended behavior if not properly configured.

```
54      /**
55       * @dev Function to initialize the contract
56       * @dev Can an only be called once by the deployer of the contract
57       * @dev The caller is responsible for ensuring that both the new owner and
    the token contract are configured correctly
58       * @param newOwner The address of the new owner of the mint contract, can
    either be an EOA or a contract
59       * @param newTokenContract The address of the token contract that is minted
60       */
61      function initialize(address newOwner, address newTokenContract)
62          external
63          onlyOwner
64      {
65          require(!initialized, "MintForwarder: contract is already
    initialized");
66          require(
67              newOwner != address(0),
68              "MintForwarder: owner is the zero address"
69          );
70          require(
71              newTokenContract != address(0),
72              "MintForwarder: tokenContract is the zero address"
73          );
74          transferOwnership(newOwner);
75          tokenContract = newTokenContract;
76          initialized = true;
77      }
```

MintForwarder.sol

```
53      /**
54       * @dev Function to initialize the contract
55       * @dev Can an only be called once by the deployer of the contract
56       * @dev The caller is responsible for ensuring that both the new owner and
    the token contract are configured correctly
57       * @param newOwner The address of the new owner of the exchange rate
    updater contract, can either be an EOA or a contract
58       * @param newTokenContract The address of the token contract whose exchange
    rate is updated
59       */
60      function initialize(address newOwner, address newTokenContract)
61          external
62          onlyOwner
63      {
64          require(
65              !initialized,
66              "ExchangeRateUpdater: contract is already initialized"
67          );
68          require(
```

```
69              newOwner != address(0),
70              "ExchangeRateUpdater: owner is the zero address"
71          );
72          require(
73              newTokenContract != address(0),
74              "ExchangeRateUpdater: tokenContract is the zero address"
75          );
76          transferOwnership(newOwner);
77          tokenContract = newTokenContract;
78          initialized = true;
79      }
```

ExchangeRateUpdater.sol

Recommendation: Add verification of smart contract accounts.

8. Lack of original address validation [Informational]

`Status: Confirmed`

Description

Multiple configuration functions of the `UnwrapTokenV1` contract lack original address verification.

```
340     /**
341     * @dev Function to update the operatorAddress
342      * @param _newOperatorAddress The new botAddress
343      */
344     function setNewOperator(address _newOperatorAddress) external onlyOwner {
345         require(_newOperatorAddress != address(0), "zero address provided");
346         operatorAddress = _newOperatorAddress;
347         emit OperatorUpdated(_newOperatorAddress);
348     }
349 
350     /**
351     * @dev Function to update the rechargeAddress
352      * @param _newRechargeAddress The new rechargeAddress
353      */
354     function setRechargeAddress(address _newRechargeAddress) external onlyOwner
    {
355         require(_newRechargeAddress != address(0), "zero address provided");
356         rechargeAddress = _newRechargeAddress;
357         emit RechargeAddressUpdated(_newRechargeAddress);
358     }
359 
360     /**
361     * @dev Function to update the ethBackAddress
362      * @param _newEthBackAddress The new ethBackAddress
363      */
364     function setEthBackAddress(address _newEthBackAddress) external onlyOwner {
365         require(_newEthBackAddress != address(0), "zero address provided");
366         ethBackAddress = _newEthBackAddress;
367         emit EthBackAddressUpdated(_newEthBackAddress);
368     }
```

UnwrapTokenV1.sol

Recommendation: Adding original address validation.

```
340       /**
341       * @dev Function to update the operatorAddress
342        * @param _newOperatorAddress The new botAddress
343        */
344       function setNewOperator(address _newOperatorAddress) external onlyOwner {
345           require(_newOperatorAddress != address(0), "zero address provided");
346   +       require(_newOperatorAddress != operatorAddress);
347           operatorAddress = _newOperatorAddress;
348           emit OperatorUpdated(_newOperatorAddress);
349       }
350
351       /**
352       * @dev Function to update the rechargeAddress
353        * @param _newRechargeAddress The new rechargeAddress
354        */
355       function setRechargeAddress(address _newRechargeAddress) external onlyOwner
      {
356           require(_newRechargeAddress != address(0), "zero address provided");
357   +       require(_newRechargeAddress != rechargeAddress);
358           rechargeAddress = _newRechargeAddress;
359           emit RechargeAddressUpdated(_newRechargeAddress);
360       }
361
362       /**
363       * @dev Function to update the ethBackAddress
364        * @param _newEthBackAddress The new ethBackAddress
365        */
366       function setEthBackAddress(address _newEthBackAddress) external onlyOwner {
367           require(_newEthBackAddress != address(0), "zero address provided");
368   +       require(_newEthBackAddress != ethBackAddress);
369           ethBackAddress = _newEthBackAddress;
370           emit EthBackAddressUpdated(_newEthBackAddress);
371       }
```

UnwrapTokenV1.sol

9. Gas optimization [Informational]

`Status: Acknowledged`

Description

`RateLimit::currentAllowance()` is used to return the caller's current allowance after replenishing the caller's allowance. However, the best practice is to check the identity of the `caller` directly, and if it is not among the `callers`, then there is no need to perform subsequent steps, which can save additional gas consumption.

```
150       /**
151        * @dev Get the current caller allowance for an account
152        * @param caller The address of the caller
153        * @return The allowance of the given caller post replenishment
154        */
155       function currentAllowance(address caller) public returns (uint256) {
156           _replenishAllowance(caller);
157           return allowances[caller];
158       }
```

RateLimit.sol

Recommendation: Add the code.

```
150      /**
151       * @dev Get the current caller allowance for an account
152       * @param caller The address of the caller
153       * @return The allowance of the given caller post replenishment
154       */
155      function currentAllowance(address caller) public returns (uint256) {
156  +       require(callers[caller]);
157          _replenishAllowance(caller);
158          return allowances[caller];
159      }
```

RateLimit.sol

10. Claim flag optimization [Informational]

`Status: Acknowledged`

Description

`WithdrawRequest.claimTime` records the exact time by the claimed request. However, in wBETH operations, it is only used by `UnwrapTokenV1#L273` for validity checking, so it can be flagged with a `bool` type instead.

```
32      struct WithdrawRequest {
33          address recipient; // user who withdraw
34          uint256 wbethAmount; //WBETH
35          uint256 ethAmount; //ETH
36          uint256 triggerTime; //user trigger time
37          uint256 claimTime; //user claim time
38          bool allocated;  //is it allocated
39      }
```

UnwrapTokenV1.sol

```
255      /**
256       * @dev claim the allocated eth
257       * @param _index the index to claim
258       * @return the eth amount
259       */
260      function claimWithdraw(uint256 _index) external whenNotPaused
261              notBlacklisted(msg.sender) returns (uint256)
262      {
263          address user = msg.sender;
264          uint256[] storage _userRequests = userWithdrawRequests[user];
265          require(_index < _userRequests.length, "Invalid index");
266
267          uint256 _allocateIndex = _userRequests[_index];
268          WithdrawRequest storage _withdrawRequest =
     withdrawRequests[_allocateIndex];
269          uint256 _ethAmount = _withdrawRequest.ethAmount;
270
271          require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime),
     "Claim time not reach");
272          require(_withdrawRequest.allocated, "Not allocated yet");
273          require(_withdrawRequest.claimTime == 0, "Already claim yet");
274          require(_getCurrentBalance() >= _ethAmount, "Not enough balance");
275
276          if (_userRequests.length > 1) {
277              _userRequests[_index] = _userRequests[_userRequests.length - 1];
278          }
```

```
279             _userRequests.pop();
280
281             _withdrawRequest.claimTime = block.timestamp;
282             _transferEth(msg.sender, _ethAmount);
283             emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
284             return _ethAmount;
285         }
```

UnwrapTokenV1.sol

Recommendation: Modify the claimTime in the `WithdrawRequest` structure to be of type bool and change the check and update sections.

Feedback: Off-chain, need to access claimTime to get the exact claim time record of the user.

11. Best Practice [Informational]

`Status: Fixed`

Description

According to the official documentation for the Solidity language, as a development specification, visibility should be placed before Modifiers.

```
314         /**
315         * @dev get need recharge eth amount
316          */
317         function getNeedRechargeEthAmount() view public returns (uint256) {
318             if (availableAllocateAmount >= needEthAmount) {
319                 return 0;
320             } else {
321                 return needEthAmount.sub(availableAllocateAmount);
322             }
323         }
324
325         /**
326         * @dev get eth balance of contract
327          */
328         function _getCurrentBalance() view internal virtual returns (uint256) {
329             return address(this).balance;
330         }
```

UnwrapTokenV1.sol

Recommendation: Optimize according to specifications

```
314         /**
315         * @dev get need recharge eth amount
316          */
317 -       function getNeedRechargeEthAmount() view public returns (uint256) {
318 +       function getNeedRechargeEthAmount() public view returns (uint256) {
319             if (availableAllocateAmount >= needEthAmount) {
320                 return 0;
321             } else {
322                 return needEthAmount.sub(availableAllocateAmount);
323             }
324         }
325
```

```
326       /**
327       * @dev get eth balance of contract
328        */
329   -    function _getCurrentBalance() view internal virtual returns (uint256) {
330   +    function _getCurrentBalance() internal view virtual returns (uint256) {
331           return address(this).balance;
332       }
```

UnwrapTokenV1.sol

12. Defined local variables not well utilized [Informational]

Status: Fixed

Description

The user variable defined within the claimWithdraw function is not fully utilized.

```
255       /**
256        * @dev claim the allocated eth
257        * @param _index the index to claim
258        * @return the eth amount
259        */
260       function claimWithdraw(uint256 _index) external whenNotPaused
261               notBlacklisted(msg.sender) returns (uint256)
262       {
263           address user = msg.sender;
264           uint256[] storage _userRequests = userWithdrawRequests[user];
265           require(_index < _userRequests.length, "Invalid index");
266
267           uint256 _allocateIndex = _userRequests[_index];
268           WithdrawRequest storage _withdrawRequest =
      withdrawRequests[_allocateIndex];
269           uint256 _ethAmount = _withdrawRequest.ethAmount;
270
271           require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime),
      "Claim time not reach");
272           require(_withdrawRequest.allocated, "Not allocated yet");
273           require(_withdrawRequest.claimTime == 0, "Already claim yet");
274           require(_getCurrentBalance() >= _ethAmount, "Not enough balance");
275
276           if (_userRequests.length > 1) {
277               _userRequests[_index] = _userRequests[_userRequests.length - 1];
278           }
279           _userRequests.pop();
280
281           _withdrawRequest.claimTime = block.timestamp;
282           _transferEth(msg.sender, _ethAmount);
283           emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
284           return _ethAmount;
285       }
```

UnwrapTokenV1.sol

Recommendation: Revise the code.

```
255       /**
256        * @dev claim the allocated eth
257        * @param _index the index to claim
```

```
258       * @return the eth amount
259       */
260      function claimWithdraw(uint256 _index) external whenNotPaused
261              notBlacklisted(msg.sender) returns (uint256)
262      {
263          address user = msg.sender;
264          uint256[] storage _userRequests = userWithdrawRequests[user];
265          require(_index < _userRequests.length, "Invalid index");
266
267          uint256 _allocateIndex = _userRequests[_index];
268          WithdrawRequest storage _withdrawRequest =
     withdrawRequests[_allocateIndex];
269          uint256 _ethAmount = _withdrawRequest.ethAmount;
270
271          require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime),
     "Claim time not reach");
272          require(_withdrawRequest.allocated, "Not allocated yet");
273          require(_withdrawRequest.claimTime == 0, "Already claim yet");
274          require(_getCurrentBalance() >= _ethAmount, "Not enough balance");
275
276          if (_userRequests.length > 1) {
277              _userRequests[_index] = _userRequests[_userRequests.length - 1];
278          }
279          _userRequests.pop();
280
281          _withdrawRequest.claimTime = block.timestamp;
282  -       _transferEth(msg.sender, _ethAmount);
283  +       _transferEth(user, _ethAmount);
284          emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
285          return _ethAmount;
286      }
```

UnwrapTokenV1.sol

13. Code optimization [Informational]

Status: Acknowledged

Description

#L299 Indentation is not standardized.

```
287      /**
288       * @dev allocated eth to every request
289       * @param _maxAllocateNum the max number
290       * @return the next allocate eth index
291       */
292      function allocate(uint256 _maxAllocateNum) external whenNotPaused
     onlyOperator returns (uint256)
293      {
294          require(needEthAmount > 0 && availableAllocateAmount > 0, "No need
     allocated or no more availableAllocateAmount ");
295          require(_maxAllocateNum <= MAX_LOOP_NUM, "Too big number > 1000");
296          require(startAllocatedEthIndex < nextIndex, "Not need allocated");
297
298          for (uint256 _reqCount = 0; _reqCount < _maxAllocateNum &&
     startAllocatedEthIndex < nextIndex &&
299
     withdrawRequests[startAllocatedEthIndex].ethAmount <= availableAllocateAmount;
```

```
300             _reqCount++
301         ) {
302             WithdrawRequest storage _withdrawRequest =
    withdrawRequests[startAllocatedEthIndex];
303             _withdrawRequest.allocated = true;
304
305             availableAllocateAmount =
    availableAllocateAmount.sub(_withdrawRequest.ethAmount);
306             needEthAmount = needEthAmount.sub(_withdrawRequest.ethAmount);
307
308             startAllocatedEthIndex++;
309         }
310         emit Allocate(operatorAddress, startAllocatedEthIndex);
311         return startAllocatedEthIndex;
312     }
```

UnwrapTokenV1.sol

Recommendation: Revise the code.

```
287     /**
288      * @dev allocated eth to every request
289      * @param _maxAllocateNum the max number
290      * @return the next allocate eth index
291      */
292     function allocate(uint256 _maxAllocateNum) external whenNotPaused
    onlyOperator returns (uint256)
293     {
294         require(needEthAmount > 0 && availableAllocateAmount > 0, "No need
    allocated or no more availableAllocateAmount ");
295         require(_maxAllocateNum <= MAX_LOOP_NUM, "Too big number > 1000");
296         require(startAllocatedEthIndex < nextIndex, "Not need allocated");
297
    +       for (uint256 _reqCount = 0; _reqCount < _maxAllocateNum &&
298 startAllocatedEthIndex < nextIndex &&
    withdrawRequests[startAllocatedEthIndex].ethAmount <= availableAllocateAmount;
    -
299 withdrawRequests[startAllocatedEthIndex].ethAmount <= availableAllocateAmount;
300             _reqCount++
301         ) {
302             WithdrawRequest storage _withdrawRequest =
    withdrawRequests[startAllocatedEthIndex];
303             _withdrawRequest.allocated = true;
304
305             availableAllocateAmount =
    availableAllocateAmount.sub(_withdrawRequest.ethAmount);
306             needEthAmount = needEthAmount.sub(_withdrawRequest.ethAmount);
307
308             startAllocatedEthIndex++;
309         }
310         emit Allocate(operatorAddress, startAllocatedEthIndex);
311         return startAllocatedEthIndex;
312     }
```

UnwrapTokenV1.sol

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.