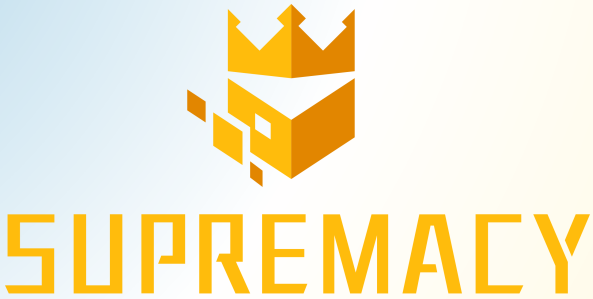
wBETH Security Audit Report

September 1st, 2023



Prepared for:
Binance

Prepared by:
Supremacy

Contents

Introduction

Given the opportunity to review the design document and related source code 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.

About Client

Wrapped BETH ("WBETH") is a special kind of BETH, and is a token created by depositing BETH into the BETH wrapper. Each WBETH represents 1 BETH (1:1 to staked ETH) plus all of its accrued ETH2.0 staking rewards starting from when WBETH's conversion rate was initialized at 1:1 on 27 Apr 2023 00:00 (UTC+0).

In other words, WBETH is reward-bearing in nature. It reflects ETH2.0 staking rewards not by growing in quantity, but by growing in value in relation to BETH. Over time, the price of WBETH will likely be worth more BETH.

Unlike BETH, WBETH accumulates staking rewards despite not being held in Binance.

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

Audit Scope

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

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

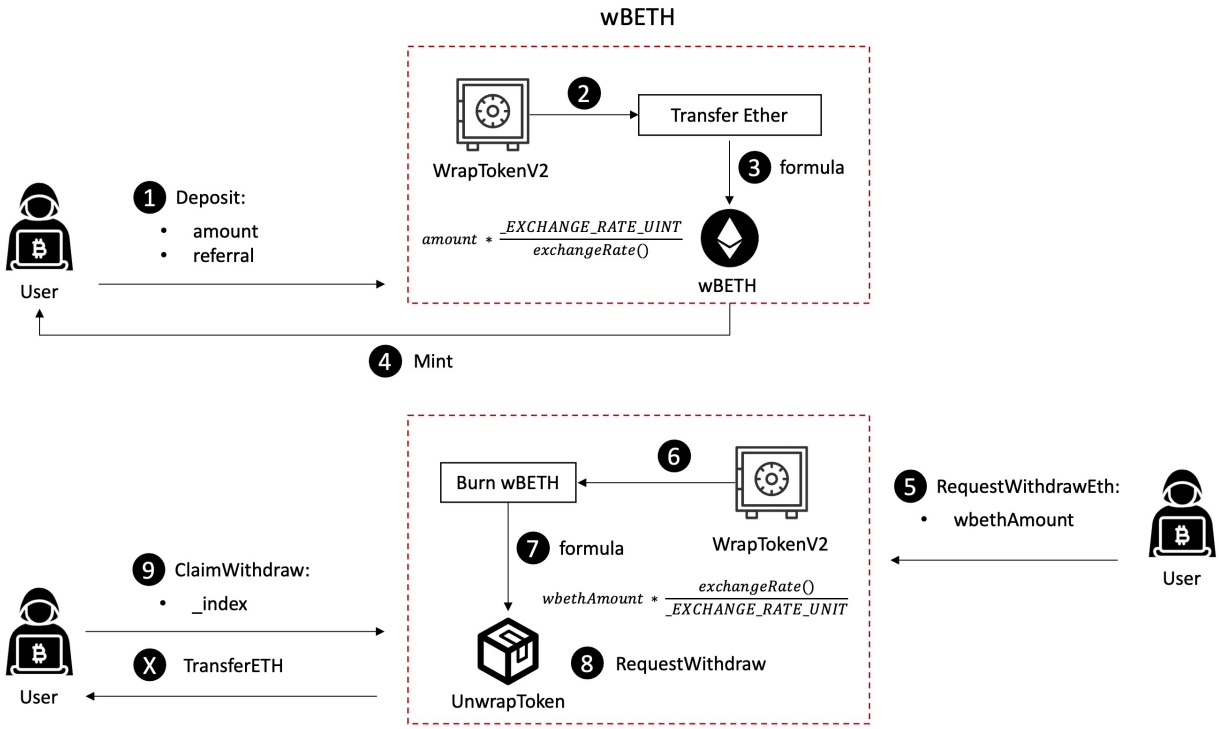
And this is the commit hash after all fixes for the issues found in the security audit have been checked in:

- https://github.com/earn-tech-git/wbeth/tree/develop_unwrap/contracts/wrapped-tokens
- Commit Hash: 2c9d21c8007e0af7c770a6fbf13cb5e1a6899d77

Changelogs

Version	Date	Description
0.1	August 29, 2023	Initial Draft
0.2	August 30, 2023	Release Candidate #1
1.0	September 1, 2023	Final Release

Threat Model



wBETH is an Liquid staking protocol, and within the scope of observable security audits its main functions are the components StakedToken, WrapToken, UnwrapToken, and ExchangeRateUpdater.

As shown above, this involves multiple interactions between a user who (wraps) his ETH into a wBETH via wBETH and a user who (unwraps) his wBETH into an ETH via wBETH. **During the audit, we assume the user could be malicious, which means all messages sent to wBETH are untrusted.**

We enumerated the attack surface based on this assumption.

About Us

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

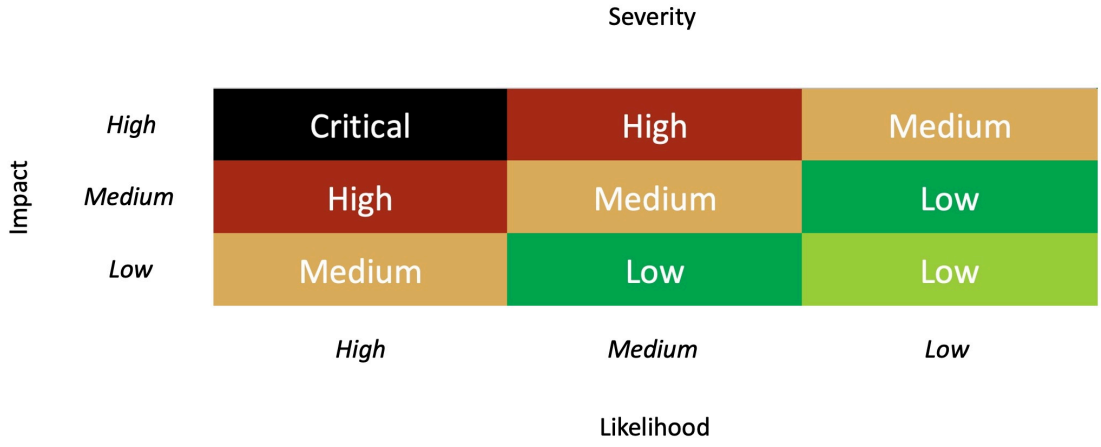
We are reachable at Telegram (https://t.me/SupremacyInc), Twitter (https://twitter.com/SupremacyHQ), or Email (contact@supremacy.email).

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.

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`

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).

```
contract WrapTokenV2BSC is StakedTokenV2 {
    /**
     * @dev ETH contract address on current chain.
     */
    address public constant _ETH_ADDRESS = 0xE7bCB9e341D546b66a46298f4893f5650a56e99E;
```

```
    /**
    * @dev UNWRAP ETH contract address on current chain.
     */
    address public constant _UNWRAP_ETH_ADDRESS = 0x5159fC6E2487828904eB1254B46365315063c86D;
```

WrapTokenV2BSC.sol

```
contract WrapTokenV2ETH is StakedTokenV2 {
    /**
     * @dev gas limit of eth transfer.
     */
    uint256 private constant _ETH_TRANSFER_GAS = 5000;

    /**
    * @dev UNWRAP ETH contract address on current chain.
     */
    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.

```
    /**
     * @dev Function to mint tokens to msg.sender
     * @param amount to mint
     */
    function mint(uint256 amount)
        external
        onlyMinters
        returns (bool)
    {
        uint256 mintingAllowedAmount = minterAllowed[msg.sender];
        require(
            amount <= mintingAllowedAmount,
            "StakedTokenV1: mint amount exceeds minterAllowance"
        );

        _mint(msg.sender, amount);

        minterAllowed[msg.sender] = mintingAllowedAmount.sub(amount);
        return true;
    }
```

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.

```
    /**
     * @notice Rescue ERC20 tokens locked up in this contract.
     * @param tokenContract ERC20 token contract address
     * @param to        Recipient address
     * @param amount    Amount to withdraw
     */
    function rescueERC20(
        IERC20 tokenContract,
        address to,
        uint256 amount
    ) external onlyRescuer {
        tokenContract.safeTransfer(to, amount);
    }
```

Rescuable.sol

Recommendation: Add the code.

```
    /**
     * @notice Rescue ERC20 tokens locked up in this contract.
     * @param tokenContract ERC20 token contract address
     * @param to        Recipient address
     * @param amount    Amount to withdraw
     */
    function rescueERC20(
        IERC20 tokenContract,
        address to,
        uint256 amount
    ) external onlyRescuer {
+       require(address(tokenContract) != _ETH_ADDRESS);
        tokenContract.safeTransfer(to, amount);
    }
```

Rescuable.sol

Feedback:

The code will be changed before deployment due to the external libraries referenced there.

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.

```
/**
 * @dev claim the allocated eth
 * @param _index the index to claim
 * @return the eth amount
 */
function claimWithdraw(uint256 _index) external whenNotPaused
        notBlacklisted(msg.sender) returns (uint256)
{
    address user = msg.sender;
    uint256[] storage _userRequests = userWithdrawRequests[user];
    require(_index < _userRequests.length, "Invalid index");

    uint256 _allocateIndex = _userRequests[_index];
    WithdrawRequest storage _withdrawRequest = withdrawRequests[_allocateIndex];
    uint256 _ethAmount = _withdrawRequest.ethAmount;

    require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime), "Claim time not reach");
    require(_withdrawRequest.allocated, "Not allocated yet");
    require(_withdrawRequest.claimTime == 0, "Already claim yet");
    require(_getCurrentBalance() >= _ethAmount, "Not enough balance");

    if (_userRequests.length > 1) {
        _userRequests[_index] = _userRequests[_userRequests.length - 1];
    }
    _userRequests.pop();

    _withdrawRequest.claimTime = block.timestamp;
    _transferEth(msg.sender, _ethAmount);
    emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
    return _ethAmount;
}
```

UnwrapTokenV1.sol

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

Recommendation: Adding Access Controls.

```
    /**
     * @dev claim the allocated eth
     * @param _index the index to claim
     * @return the eth amount
     */
    function claimWithdraw(uint256 _index) external whenNotPaused
            notBlacklisted(msg.sender) returns (uint256)
    {
        address user = msg.sender;
        uint256[] storage _userRequests = userWithdrawRequests[user];
        require(_index < _userRequests.length, "Invalid index");

        uint256 _allocateIndex = _userRequests[_index];
        WithdrawRequest storage _withdrawRequest = withdrawRequests[_allocateIndex];
        uint256 _ethAmount = _withdrawRequest.ethAmount;

+       require(_withdrawRequest.recipient == user, "Wrong recipient");
        require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime), "Claim time not reach");
        require(_withdrawRequest.allocated, "Not allocated yet");
        require(_withdrawRequest.claimTime == 0, "Already claim yet");
        require(_getCurrentBalance() >= _ethAmount, "Not enough balance");

        if (_userRequests.length > 1) {
```

```
            _userRequests[_index] = _userRequests[_userRequests.length - 1];
        }
        _userRequests.pop();

        _withdrawRequest.claimTime = block.timestamp;
        _transferEth(msg.sender, _ethAmount);
        emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
        return _ethAmount;
    }
```

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.

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

        // msg.value and exchangeRate are all scaled by 1e18
        uint256 wBETHAmount = msg.value.mul(_EXCHANGE_RATE_UNIT).div(exchangeRate());

        _mint(msg.sender, wBETHAmount);

        emit DepositEth(msg.sender, msg.value, wBETHAmount, referral);
    }
```

WrapTokenV2ETH.sol

The minting calculation formula is as follows:

$$_EXCHANGE_RATE_UNIT = 1e18$$

$$exchangeRate() = 1e18$$

$$msg.value * _EXCHANGE_RATE_UNIT / exchangeRate()$$

```
    /**
     * @dev Function to withdraw wBETH for eth
     * @param wbethAmount The wBETH amount
     */
    function requestWithdrawEth(uint256 wbethAmount) external {
        require(wbethAmount > 0, "zero wBETH amount");

        // msg.value and exchangeRate are all scaled by 1e18
        uint256 ethAmount = wbethAmount.mul(exchangeRate()).div(_EXCHANGE_RATE_UNIT);
        _burn(wbethAmount);
        IUnwrapTokenV1(_UNWRAP_ETH_ADDRESS).requestWithdraw(msg.sender, wbethAmount, ethAmount);
        emit RequestWithdrawEth(msg.sender, wbethAmount, ethAmount);
    }
```

WrapTokenV2ETH.sol

The withdrawal calculation formula is as follows:

$$exchangeRate() = 1$$

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

$$wbethAmount * exchangeRate() / \_EXCHANGE\_RATE\_UNIT$$

Recommendation: Strict control of exchange rate adjustments.

Informational

---

7. Lack of isContract validation **[Informational]**

   Status: Confirmed

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

```
    /**
     * @dev Function to initialize the contract
     * @dev Can an only be called once by the deployer of the contract
     * @dev The caller is responsible for ensuring that both the new owner and the token contract are configured
correctly
     * @param newOwner The address of the new owner of the mint contract, can either be an EOA or a contract
     * @param newTokenContract The address of the token contract that is minted
     */
    function initialize(address newOwner, address newTokenContract)
        external
        onlyOwner
    {
        require(!initialized, "MintForwarder: contract is already initialized");
        require(
            newOwner != address(0),
            "MintForwarder: owner is the zero address"
        );
        require(
            newTokenContract != address(0),
            "MintForwarder: tokenContract is the zero address"
        );
        transferOwnership(newOwner);
        tokenContract = newTokenContract;
        initialized = true;
    }
```

MintForwarder.sol

```
    /**
     * @dev Function to initialize the contract
     * @dev Can an only be called once by the deployer of the contract
     * @dev The caller is responsible for ensuring that both the new owner and the token contract are configured
correctly
     * @param newOwner The address of the new owner of the exchange rate updater contract, can either be an EOA or a
contract
     * @param newTokenContract The address of the token contract whose exchange rate is updated
     */
    function initialize(address newOwner, address newTokenContract)
        external
        onlyOwner
    {
        require(
            !initialized,
            "ExchangeRateUpdater: contract is already initialized"
        );
        require(
            newOwner != address(0),
            "ExchangeRateUpdater: owner is the zero address"
        );
        require(
            newTokenContract != address(0),
            "ExchangeRateUpdater: tokenContract is the zero address"
        );
        transferOwnership(newOwner);
        tokenContract = newTokenContract;
        initialized = true;
    }
```

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.

```
    /**
    * @dev Function to update the operatorAddress
     * @param _newOperatorAddress The new botAddress
     */
    function setNewOperator(address _newOperatorAddress) external onlyOwner {
        require(_newOperatorAddress != address(0), "zero address provided");
        operatorAddress = _newOperatorAddress;
        emit OperatorUpdated(_newOperatorAddress);
    }

    /**
    * @dev Function to update the rechargeAddress
     * @param _newRechargeAddress The new rechargeAddress
     */
    function setRechargeAddress(address _newRechargeAddress) external onlyOwner {
        require(_newRechargeAddress != address(0), "zero address provided");
        rechargeAddress = _newRechargeAddress;
        emit RechargeAddressUpdated(_newRechargeAddress);
    }

    /**
    * @dev Function to update the ethBackAddress
     * @param _newEthBackAddress The new ethBackAddress
     */
    function setEthBackAddress(address _newEthBackAddress) external onlyOwner {
        require(_newEthBackAddress != address(0), "zero address provided");
        ethBackAddress = _newEthBackAddress;
        emit EthBackAddressUpdated(_newEthBackAddress);
    }
```

UnwrapTokenV1.sol

Recommendation: Adding original address validation.

```
    /**
    * @dev Function to update the operatorAddress
     * @param _newOperatorAddress The new botAddress
     */
    function setNewOperator(address _newOperatorAddress) external onlyOwner {
        require(_newOperatorAddress != address(0), "zero address provided");
+       require(_newOperatorAddress != operatorAddress);

        operatorAddress = _newOperatorAddress;
        emit OperatorUpdated(_newOperatorAddress);
    }

    /**
    * @dev Function to update the rechargeAddress
     * @param _newRechargeAddress The new rechargeAddress
     */
    function setRechargeAddress(address _newRechargeAddress) external onlyOwner {
        require(_newRechargeAddress != address(0), "zero address provided");
+       require(_newRechargeAddress != rechargeAddress);

        rechargeAddress = _newRechargeAddress;
        emit RechargeAddressUpdated(_newRechargeAddress);
    }

    /**
    * @dev Function to update the ethBackAddress
     * @param _newEthBackAddress The new ethBackAddress
     */
    function setEthBackAddress(address _newEthBackAddress) external onlyOwner {
        require(_newEthBackAddress != address(0), "zero address provided");
```

```
+       require(_newEthBackAddress != ethBackAddress);

        ethBackAddress = _newEthBackAddress;
        emit EthBackAddressUpdated(_newEthBackAddress);
    }
```

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.

```
    /**
     * @dev Get the current caller allowance for an account
     * @param caller The address of the caller
     * @return The allowance of the given caller post replenishment
     */
    function currentAllowance(address caller) public returns (uint256) {
        _replenishAllowance(caller);
        return allowances[caller];
    }
```

RateLimit.sol

Recommendation: Add the code.

```
    /**
     * @dev Get the current caller allowance for an account
     * @param caller The address of the caller
     * @return The allowance of the given caller post replenishment
     */
    function currentAllowance(address caller) public returns (uint256) {
+       require(callers[caller]);
        _replenishAllowance(caller);
        return allowances[caller];
    }
```

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.

```
    struct WithdrawRequest {
        address recipient; // user who withdraw
        uint256 wbethAmount; //WBETH
        uint256 ethAmount; //ETH
        uint256 triggerTime; //user trigger time
        uint256 claimTime; //user claim time
        bool allocated;  //is it allocated
    }
```

UnwrapTokenV1.sol

```
    /**
     * @dev claim the allocated eth
     * @param _index the index to claim
     * @return the eth amount
     */
    function claimWithdraw(uint256 _index) external whenNotPaused
            notBlacklisted(msg.sender) returns (uint256)
    {
```

```
        address user = msg.sender;
        uint256[] storage _userRequests = userWithdrawRequests[user];
        require(_index < _userRequests.length, "Invalid index");

        uint256 _allocateIndex = _userRequests[_index];
        WithdrawRequest storage _withdrawRequest = withdrawRequests[_allocateIndex];
        uint256 _ethAmount = _withdrawRequest.ethAmount;

        require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime), "Claim time not reach");
        require(_withdrawRequest.allocated, "Not allocated yet");
        require(_withdrawRequest.claimTime == 0, "Already claim yet");
        require(_getCurrentBalance() >= _ethAmount, "Not enough balance");

        if (_userRequests.length > 1) {
            _userRequests[_index] = _userRequests[_userRequests.length - 1];
        }
        _userRequests.pop();

        _withdrawRequest.claimTime = block.timestamp;
        _transferEth(msg.sender, _ethAmount);
        emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
        return _ethAmount;
    }
```

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 monitoring tools 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.

```
    /**
    * @dev get need recharge eth amount
     */
    function getNeedRechargeEthAmount() view public returns (uint256) {
        if (availableAllocateAmount >= needEthAmount) {
            return 0;
        } else {
            return needEthAmount.sub(availableAllocateAmount);
        }
    }

    /**
    * @dev get eth balance of contract
     */
    function _getCurrentBalance() view internal virtual returns (uint256) {
        return address(this).balance;
    }
```

UnwrapTokenV1.sol

Recommendation: Optimize according to specifications

```
    /**
    * @dev get need recharge eth amount
     */
-    function getNeedRechargeEthAmount() view public returns (uint256) {
+    function getNeedRechargeEthAmount() public view returns (uint256) {
        if (availableAllocateAmount >= needEthAmount) {
            return 0;
        } else {
            return needEthAmount.sub(availableAllocateAmount);
        }
    }

    /**
    * @dev get eth balance of contract
```

```
     */
-    function _getCurrentBalance() view internal virtual returns (uint256) {
+    function _getCurrentBalance() internal view virtual returns (uint256) {
         return address(this).balance;
     }
```

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.

```
    /**
     * @dev claim the allocated eth
     * @param _index the index to claim
     * @return the eth amount
     */
    function claimWithdraw(uint256 _index) external whenNotPaused
            notBlacklisted(msg.sender) returns (uint256)
    {
        address user = msg.sender;
        uint256[] storage _userRequests = userWithdrawRequests[user];
        require(_index < _userRequests.length, "Invalid index");

        uint256 _allocateIndex = _userRequests[_index];
        WithdrawRequest storage _withdrawRequest = withdrawRequests[_allocateIndex];
        uint256 _ethAmount = _withdrawRequest.ethAmount;

        require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime), "Claim time not reach");
        require(_withdrawRequest.allocated, "Not allocated yet");
        require(_withdrawRequest.claimTime == 0, "Already claim yet");
        require(_getCurrentBalance() >= _ethAmount, "Not enough balance");

        if (_userRequests.length > 1) {
            _userRequests[_index] = _userRequests[_userRequests.length - 1];
        }
        _userRequests.pop();

        _withdrawRequest.claimTime = block.timestamp;
        _transferEth(msg.sender, _ethAmount);
        emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
        return _ethAmount;
    }
```

UnwrapTokenV1.sol

Recommendation: Modify code.

```
    /**
     * @dev claim the allocated eth
     * @param _index the index to claim
     * @return the eth amount
     */
    function claimWithdraw(uint256 _index) external whenNotPaused
            notBlacklisted(msg.sender) returns (uint256)
    {
        address user = msg.sender;
        uint256[] storage _userRequests = userWithdrawRequests[user];
        require(_index < _userRequests.length, "Invalid index");

        uint256 _allocateIndex = _userRequests[_index];
        WithdrawRequest storage _withdrawRequest = withdrawRequests[_allocateIndex];
        uint256 _ethAmount = _withdrawRequest.ethAmount;

        require(block.timestamp >= _withdrawRequest.triggerTime.add(lockTime), "Claim time not reach");
        require(_withdrawRequest.allocated, "Not allocated yet");
        require(_withdrawRequest.claimTime == 0, "Already claim yet");
        require(_getCurrentBalance() >= _ethAmount, "Not enough balance");

        if (_userRequests.length > 1) {
            _userRequests[_index] = _userRequests[_userRequests.length - 1];
        }
```

```
        _userRequests.pop();

        _withdrawRequest.claimTime = block.timestamp;
-        _transferEth(msg.sender, _ethAmount);
+        _transferEth(user, _ethAmount);
        emit ClaimWithdraw(user, _ethAmount, _allocateIndex);
        return _ethAmount;
    }
```

UnwrapTokenV1.sol

---

13. Code optimization **[Informational]**

    Status: Acknowledged

    Description: #L299 Indentation is not standardized.

```
    /**
     * @dev allocated eth to every request
     * @param _maxAllocateNum the max number
     * @return the next allocate eth index
     */
    function allocate(uint256 _maxAllocateNum) external whenNotPaused onlyOperator returns (uint256)
    {
        require(needEthAmount > 0 && availableAllocateAmount > 0, "No need allocated or no more availableAllocateAmount
");
        require(_maxAllocateNum <= MAX_LOOP_NUM, "Too big number > 1000");
        require(startAllocatedEthIndex < nextIndex, "Not need allocated");

        for (uint256 _reqCount = 0; _reqCount < _maxAllocateNum && startAllocatedEthIndex < nextIndex &&
                            withdrawRequests[startAllocatedEthIndex].ethAmount <= availableAllocateAmount;
            _reqCount++
        ) {
            WithdrawRequest storage _withdrawRequest = withdrawRequests[startAllocatedEthIndex];
            _withdrawRequest.allocated = true;

            availableAllocateAmount = availableAllocateAmount.sub(_withdrawRequest.ethAmount);
            needEthAmount = needEthAmount.sub(_withdrawRequest.ethAmount);

            startAllocatedEthIndex++;
        }
        emit Allocate(operatorAddress, startAllocatedEthIndex);
        return startAllocatedEthIndex;
    }
```

UnwrapTokenV1.sol

Recommendation: Modify code.

```
    /**
     * @dev allocated eth to every request
     * @param _maxAllocateNum the max number
     * @return the next allocate eth index
     */
    function allocate(uint256 _maxAllocateNum) external whenNotPaused onlyOperator returns (uint256)
    {
        require(needEthAmount > 0 && availableAllocateAmount > 0, "No need allocated or no more availableAllocateAmount
");
        require(_maxAllocateNum <= MAX_LOOP_NUM, "Too big number > 1000");
        require(startAllocatedEthIndex < nextIndex, "Not need allocated");

+       for (uint256 _reqCount = 0; _reqCount < _maxAllocateNum && startAllocatedEthIndex < nextIndex &&
withdrawRequests[startAllocatedEthIndex].ethAmount <= availableAllocateAmount;
-                           withdrawRequests[startAllocatedEthIndex].ethAmount <= availableAllocateAmount;
            _reqCount++
        ) {
            WithdrawRequest storage _withdrawRequest = withdrawRequests[startAllocatedEthIndex];
            _withdrawRequest.allocated = true;

            availableAllocateAmount = availableAllocateAmount.sub(_withdrawRequest.ethAmount);
            needEthAmount = needEthAmount.sub(_withdrawRequest.ethAmount);

            startAllocatedEthIndex++;
        }
```

```
        emit Allocate(operatorAddress, startAllocatedEthIndex);
        return startAllocatedEthIndex;
    }
```

UnwrapTokenV1.sol

Disclaimer

This 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 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.