



Cyberscope

Audit Report

Tea-Fi

December 2024

Files TeaStaking.sol, SignatureHandler.sol

Audited by © cyberscope

Table of Contents

Table of Contents	1
Risk Classification	3
Review	4
Audit Updates	4
Source Files	4
Audit Scope	6
Overview	7
Stake Functionality	7
Claim Functionality	7
Unstake Functionality	7
Withdraw Functionality	8
Rewards Distribution Functionality	8
DEFAULT_ADMIN_ROLE Functionalities	8
Findings Breakdown	9
Diagnostics	10
CR - Code Repetition	11
Description	11
Recommendation	12
IHD - Incorrect Hash Declaration	13
Description	13
Recommendation	13
IVT - Incorrect Vesting Transfer	14
Description	14
Recommendation	14
Team Update	15
ODV - Operator Dependency Vulnerability	16
Description	16
Recommendation	17
Team Update	17
PTAI - Potential Transfer Amount Inconsistency	18
Description	18
Recommendation	19
Team Update	20
OCTD - Transfers Contract's Tokens	21
Description	21
Recommendation	21
Team Update	22
UVC - Unfavorable VIP Conditions	23
Description	23

Recommendation	24
Team Update	24
Functions Analysis	26
Inheritance Graph	28
Flow Graph	29
Summary	30
Disclaimer	31
About Cyberscope	32

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation:** This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation:** This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical:** Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium:** Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor:** Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative:** Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Audit Updates

Initial Audit	26 Sep 2024 https://github.com/cyberscope-io/audits/blob/main/1-tea/v1/audit.pdf
Corrected Phase 2	25 Oct 2024 https://github.com/cyberscope-io/audits/blob/main/1-tea/v2/audit.pdf
Corrected Phase 3	30 Oct 2024 https://github.com/cyberscope-io/audits/blob/main/1-tea/v3/audit.pdf
Corrected Phase 4	09 Dec 2024

Source Files

Filename	SHA256
TeaStaking.sol	0bc255d619c3a1535f9a5a3f02805d36184170ba9b4a8e8d721c19a0cc1b1afd
SignatureHandler.sol	f75213fcc9cfa5e08a8e614bb957d830c41a77767c950bcc2f18aaa34625a559
interfaces/ZeroAddressError.sol	275337ccc2d94e408edb2643adc10270df4e2a10d98207974cd95889b8aec066
interfaces/ITokenPermitSignatureDetails.sol	589f90f442d67075af7d989dfd63db0352f92cc2d2bfcc78d16268834f749e01
interfaces/ITeaVesting.sol	fada2cede0083f79f8a58be620e46c32fbe350f5ab46dc9c340007e3cd45190e

interfaces/ITeaStaking.sol	e6200ba30ed9572fe1fae8c710022059932 00066f311db8309181aba0d12a1a8
interfaces/IStruct.sol	6c461aa9f2e7944000d8e6600db6b78b9e eb78741e7294e06197dc27f3a61bd5
interfaces/IAllowanceTransfer.sol	0818719d55731ba54662dea4e4e6edbffb 57f83c7de93bbf4f4c8eee1ddc9f73
components/Permitable.sol	3caf14b467727408b5d632acfd54d21cd4 088415d004e2f06ad811f38facfdc8

Audit Scope

The staking process heavily depends on the `TeaVesting` contract, as it plays a crucial role not only in determining users' vesting allocations for staking but also in completing the withdrawal functionality. The current contract calls and interacts with the TeaVesting contract to validate vesting amounts and to facilitate the unlocking of rewards during the withdrawal process. Any inaccurate data or malfunction within the TeaVesting contract can halt the correct execution of this contract, potentially leading to blocked withdrawals or incorrect reward calculations. Therefore, it is highly recommended that the team deploy and utilize a TeaVesting contract that has been thoroughly audited to ensure the overall integrity of the system and prevent any disruptions or vulnerabilities in the staking and withdrawal processes.

Overview

The `TeaStaking` contract enables users to stake Tea tokens and approved presale tokens, offering reward allocations and stake management functionality. It facilitates staking with options for Tea tokens or vested allocations from the `TeaVesting` contract. The contract integrates cooldown mechanisms for withdrawals, ensures predictable staking rewards, and supports robust security features.

Stake Functionality

The `stake` function allows users to stake Tea tokens or utilize their vested allocations from the `TeaVesting` contract. Users can stake multiple tokens in a single transaction, provided the tokens are valid and not already staked. When a user's stake reaches a minimum threshold (`vipAmount`), VIP status is granted, locking the tokens for one year. Each stake is recorded with a unique ID, updating the total staked tokens and maintaining detailed records of the user's stakes.

Claim Functionality

The `claim` function allows users to claim rewards associated with their stakes. Users can provide the IDs of their stakes and the calculated rewards with loyalty bonuses. The function verifies the provided data using off-chain signatures, ensuring that calculations meet the required thresholds. If successful, rewards are added to the user's reward debt, and the corresponding Tea tokens are transferred. The process ensures transparent and secure reward claiming while preventing over-claiming.

Unstake Functionality

The `unstake` function initiates the unstaking process for users' stakes, enabling them to begin withdrawing their staked tokens. A two-week cooldown period is applied after unstaking, ensuring a structured timeline for token withdrawals. During this period, staked tokens become "available" but cannot yet be withdrawn. The function updates staking records, adjusts the total staked token count, and emits relevant events.

Withdraw Functionality

The `withdraw` function lets users retrieve their staked tokens and accumulated rewards after meeting claim cooldown requirements. The function checks the cooldown period, ensures no VIP locks are active, and transfers the tokens and rewards to the user. It updates staking records and emits a withdrawal event, ensuring a secure and transparent withdrawal process.

Rewards Distribution Functionality

The `updateRewardPerShare` function governs reward distribution. It calculates and updates rewards per share based on the total staked tokens and elapsed time. By maintaining an up-to-date rewards-per-share value, the contract ensures that rewards are equitably distributed among all users.

DEFAULT_ADMIN_ROLE Functionalities

The `DEFAULT_ADMIN_ROLE` enables certain administrative privileges, such as initializing staking with a specific allocation and reward distribution start time. The admin can also execute emergency withdrawals of Tea tokens from the contract, excluding staked amounts. This administrative flexibility enhances the contract's overall resilience and maintains security in emergency situations.

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	1	6	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CR	Code Repetition	Acknowledged
●	IHD	Incorrect Hash Declaration	Unresolved
●	IVT	Incorrect Vesting Transfer	Acknowledged
●	ODV	Operator Dependency Vulnerability	Acknowledged
●	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
●	OCTD	Transfers Contract's Tokens	Acknowledged
●	UVC	Unfavorable VIP Conditions	Acknowledged

CR - Code Repetition

Criticality	Minor / Informative
Location	TeaStaking.sol#L433
Status	Acknowledged

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
function claim(UnstakeParam calldata claimStruct) external {
    address user = _msgSender();
    (bool success, string memory errorReason) = _verifySignature(
        user,
        claimStruct,
        false
    );
    require(success, errorReason);

    ...
    ...

    function _unstake(
        uint256[] memory ids,
        uint256[] calldata rewardsWithLoyalty
    ) private {
        if (ids.length != rewardsWithLoyalty.length) revert
        InvalidArrayLengths();
        address user = _msgSender();
        uint256 length = ids.length;
        uint256 amountToClaim = 0;

        ...
    }
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

IHD - Incorrect Hash Declaration

Criticality	Minor / Informative
Location	SignatureHandler.sol#L15
Status	Unresolved

Description

The contract uses a hardcoded value for the `CLAIM_TYPEHASH` that does not match the actual `keccak256` hash of the provided string `"unstake(UnstakeParam calldata _params)"`. This mismatch may lead to failures in functionality relying on the hash, such as verifying signatures or validating data integrity. The issue introduces a risk of operational errors, as the hardcoded value might not align with expected calculations or external dependencies.

```
/// @dev keccak256("unstake(UnstakeParam calldata _params)")
bytes32 public constant CLAIM_TYPEHASH =
0x15aa6b515a0ec6a27e4e54b358f3569fc0dc1bf085748dc59491c935e9111f4a;
```

Recommendation

It is recommended to ensure that the `CLAIM_TYPEHASH` value is correctly derived from the `keccak256` hash of the intended string. This can be achieved by recalculating the hash programmatically and validating it against the expected value to prevent inconsistencies. Additionally, any dependencies relying on this value should be updated accordingly to maintain seamless functionality.

IVT - Incorrect Vesting Transfer

Criticality	Minor / Informative
Location	TeaStaking.sol#L307
Status	Acknowledged

Description

The contract, during the execution of the `withdraw` function, calls the `transferOwnerOnChain` function of the vesting contract to transfer the ownership of the vesting to the user. However, the usage of the `user` parameter is misleading since it does not align with the actual implementation of the vesting contract. The vesting contract may be designed to transfer ownership to the zero address (`ZERO_ADDRESS`), irrespective of the provided `user` address. This discrepancy indicates that the current code does not reflect the intended functionality and may lead to unexpected behavior or errors in the ownership transfer process.

```
function withdraw(uint256[] memory ids) external
nonReentrant {
    address user = _msgSender();
    uint256 length = ids.length;

    for (uint256 i = 0; i < length; ++i) {
        ...
    } else {
        teaVesting.claim(token, user);
        teaVesting.transferOwnerOnChain(token,
address(this), user);
    }
    emit Withdrawal(user, token, _id,
_availableTokens);
}
```

Recommendation

To enhance clarity and ensure accurate functionality, it's recommended to replace the `user` parameter with the `ZERO_ADDRESS` in the `transferOwnerOnChain` function. This adjustment aligns with the intended behavior of the vesting contract, ensuring

ownership transfer accurately reflects the contract's purpose and avoids potential inconsistencies or unintended misuse of the function.

Although the current implementation functions correctly, using the `user` parameter is unnecessary and may introduce ambiguity. Replacing it with the `ZERO_ADDRESS` both improves readability and aligns the contract with standard ownership transfer practices.

Team Update

The team has acknowledged that this is not a security issue and states:

In the Vesting contract there is already a check during the ownership transfer, that if this is the user's address, then the next owner is a zero address.

ODV - Operator Dependency Vulnerability

Criticality	Minor / Informative
Location	TeaStaking.sol#L272,280
Status	Acknowledged

Description

The contract contains the `unstake` function that must be called before the `withdraw` function, allowing users to claim their tokens. However, the `unstake` function requires a signature verification from an operator. If the operator is unable to sign the transaction, due to issues like a backend failure or unavailability, the required signature cannot be obtained, and users will be unable to proceed with unstaking and withdrawing their tokens. This dependency will lock users out of their funds under those conditions.

```
function unstake(UnstakeParam calldata unstakeParams)
external {
    (bool success, string memory errorReason) =
    _verifyUnstakeSignature(_msgSender(), unstakeParams);
    require(success, errorReason);

    _unstake(unstakeParams.ids,
unstakeParams.rewardsWithLoyalty);
}

function withdraw(uint256[] memory ids) external
nonReentrant {
    address user = _msgSender();
    uint256 length = ids.length;

    ...

    Stake storage userStake = stakes[_id];
    if (userStake.availableTokens == 0) {
        revert NeedToUnstakeFirst(_id);
    }
    ...
}
```

Recommendation

It is recommended to include an emergency withdrawal function that allows users to claim their staked amounts without requiring a signature verification in cases where the operator is unavailable. This will ensure that users have a fallback mechanism to access their funds in emergency situations, improving the resilience and reliability of the contract.

Team Update

The team has acknowledged that this is not a security issue and states:

The backend is designed to be highly reliable, ensuring that operator downtime is extremely rare.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	TeaStaking.sol#L179 Permitable.sol#L71
Status	Acknowledged

Description

The `safeTransferFrom` function is used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
function stake(address[] calldata tokens, uint256[]  
calldata amounts, OffChainStruct[] calldata offChainData)  
public {  
    bool active = startDate < block.timestamp &&  
block.timestamp <= endDate;  
    if (!active) revert StakingNotActive();  
    if (tokens.length != amounts.length || amounts.length  
!= offChainData.length) revert InvalidArrayLengths();  
  
    address user = _msgSender();  
    uint256 length = tokens.length;  
  
    for (uint256 i = 0; i < length; ++i) {  
        address _token = tokens[i];  
        uint256 _amount = amounts[i];  
  
        if (_amount == 0) revert NoZeroAmount();  
        if (!checkTokenValidity(_token)) revert  
OnlyValidToken();  
        if (_isTeaToken(_token)) {  
            _receivePayment(_token, address(this),  
_amount);  
            ...  
        }  
    }  
}
```

```
function _receivePayment(address token, address to,  
uint256 amount) internal {  
    if (amount > 0) {  
        permit2.transferFrom(_msgSender(), to,  
uint160(amount), token);  
    }  
}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance  
Before Transfer
```

Team Update

The team has acknowledged that this is not a security issue and states:

The TeaStaking contract was designed to be used with our tokens - TEA and presale tokens. They do not have transfer fees, otherwise the team would have included this logic in the staking process.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	TeaStaking.sol#L143
Status	Acknowledged

Description

The `DEFAULT_ADMIN_ROLE` role has the authority to claim all the balance of the tea tokens on the contract. The `DEFAULT_ADMIN_ROLE` role may take advantage of it by calling the `emergencyWithdraw` function.

```
function emergencyWithdraw() external {
    _checkRole(DEFAULT_ADMIN_ROLE, msg.sender);
    IERC20(teaToken).safeTransfer(msg.sender,
    IERC20(teaToken).balanceOf(address(this)));
}
```

Recommendation

The team should carefully manage the private keys of the `DEFAULT_ADMIN_ROLE's` account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

Team Update

The team has acknowledged that this is not a security issue and states:

The owner is a multisig contract where the administrators are 3 people and to implement a transaction, at least 2 signatures are needed, which minimizes centralization. This function doesn't affect the main logic, it was added only for tokens not to be locked in smart-contract by mistake.

UVC - Unfavorable VIP Conditions

Criticality	Minor / Informative
Location	TeaStaking.sol#L213
Status	Acknowledged

Description

The contract assigns VIP status to users who stake an amount greater than or equal to the `VIP_AMOUNT`. However, instead of providing additional benefits or rewards to these VIP users, the contract imposes an additional lock duration of one year on their staked tokens. As a result, users who stake large amounts and attain VIP status will be unable to unstake or withdraw their tokens until the lock period has passed, which could be a disincentive for high-value stakers.


```
function stake(address[] calldata _tokens, uint256[]  
calldata _amounts, OffChainStruct calldata _offChain)  
    external  
{  
    ...  
  
    bool _vip = false;  
    uint256 _lockedPeriod = 0;  
    if (_amount >= VIP_AMOUNT) {  
        _vip = true;  
        _lockedPeriod = block.timestamp + ONE_YEAR;  
    }  
    ...  
}  
  
function _unstake(uint256[] memory _ids, uint256[] calldata  
_rewardsWithLoyalty) private {  
    ...  
    address user = _msgSender();  
  
    for (uint256 i = 0; i < _ids.length; i++) {  
        ...  
  
        Stake storage userStake = stakes[_id];  
        if (userStake.vip && !(block.timestamp > endDate +  
30 days)) {  
            if (block.timestamp < userStake.lockedPeriod)  
revert LockedPeriodNotPassed(_id);  
        }  
        ...  
    }  
}
```

Recommendation

It is recommended to reconsider the design of the VIP status and its associated conditions. Instead of imposing a lengthy lock period, the contract should provide additional rewards or incentives for VIP users, such as higher reward rates or exclusive benefits. This change will make the VIP status more attractive and encourage users to stake larger amounts without the concern of being locked out from their funds for an extended period.

Team Update

The team has acknowledged that this is not a security issue and states:

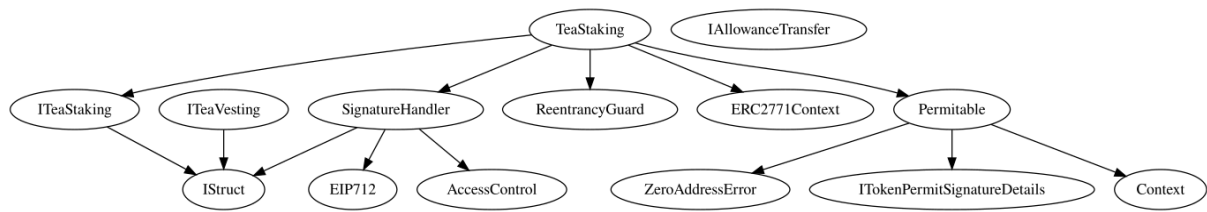
In our system VIP accounts will get discounts on transaction fees and different amount of rewards which will be calculated by our back-end.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
TeaStaking	Implementation	ITeaStaking, ReentrancyGuard, SignatureHandler, ERC2771Context, Permitable		
		Public	✓	ERC2771Context SignatureHandler Permitable
	initializeStaking	External	✓	-
	emergencyWithdraw	External	✓	-
	stake	External	✓	-
	stake	Public	✓	-
	stake	Public	✓	-
	claim	External	✓	-
	unstake	External	✓	-
	withdraw	External	✓	nonReentrant
	getUserIds	Public		-
	getPendingRewards	Public		-
	getTotalUserStakedTokens	Public		-
	updateRewardPerShare	Public	✓	-
	checkTokenValidity	Public		-

	_unstake	Private	✓	
	_isIdValid	Private		
	_harvest	Private		
	_removeFromSystem	Private	✓	
	_isTeaToken	Private		
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
SignatureHandler	Implementation	EIP712, AccessControl, IStruct		
		Public	✓	EIP712
	_verifySignature	Internal	✓	
	_verifySignature	Internal	✓	
	hashTypedDataV4	External		-

Inheritance Graph



Flow Graph



Summary

The Tea-Fi Staking contract facilitates staking and reward distribution for Tea and presale tokens. This audit reviews security vulnerabilities, business logic issues, and potential optimizations to ensure safe and efficient operation.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security. Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

cyberscope.io