



Cyberscope

Audit Report

Tea-Fi

May 2025

Files TeoFiMultiSend.sol

Audited by © cyberscope

Table of Contents

| | |
|--|-----------|
| Table of Contents | 1 |
| Risk Classification | 2 |
| Review | 3 |
| Audit Updates | 3 |
| Source Files | 3 |
| Overview | 4 |
| TeaFiMultiSend | 4 |
| Key Capabilities | 4 |
| Permit Integration | 4 |
| Operator-Based Authorization | 4 |
| Role-Based Access Control | 5 |
| Meta-Transaction Support | 5 |
| Architecture and Design Considerations | 5 |
| Roles | 5 |
| Admins | 5 |
| Operators | 6 |
| Users | 6 |
| Findings Breakdown | 7 |
| Diagnostics | 8 |
| AAO - Accumulated Amount Overflow | 9 |
| Description | 9 |
| Recommendation | 9 |
| Team Update | 9 |
| CCR - Contract Centralization Risk | 10 |
| Description | 10 |
| Recommendation | 10 |
| Team Update | 10 |
| PTAI - Potential Transfer Amount Inconsistency | 11 |
| Description | 11 |
| Recommendation | 13 |
| Team Update | 13 |
| Functions Analysis | 14 |
| Inheritance Graph | 16 |
| Summary | 17 |
| Disclaimer | 18 |
| About Cyberscope | 19 |

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 | 07 May 2025 https://github.com/cyberscope-io/audits/blob/main/tea-fi/v1/multiSend.pdf |
| Corrected Phase 2 | 27 May 2025 |

Source Files

| Filename | SHA256 |
|--------------------|--|
| TeaFiMultiSend.sol | 4dbceb16a52cff39090f8c9dd8daf23c680 06c348c6ce58c8fce15621824452c |

Overview

TeaFiMultiSend

The `TeaFiMultiSend` smart contract is an extensible design for executing both **single and batch token or ETH transfers** within the Tea-Fi ecosystem. It supports transfers that include optional **fee payments**, and leverages **permit-based approvals** using EIP-2612, DAI-style permits, and **Uniswap's Permit2** standard.

Key Capabilities

The contract allows users to send ERC-20 tokens or ETH to **a single recipient** or to **multiple recipients** simultaneously. It supports complex use cases involving fee deduction, by allowing **up to three different fee payments** per batch operation. These fees are collected and forwarded to a predefined **multisig wallet**, which acts as the central treasury for the ecosystem. This fee mechanism is controlled by signatures provided by an authorized operator, ensuring that fees are only accepted if explicitly approved off-chain and cryptographically signed.

Permit Integration

To reduce the friction of on-chain approvals and save gas costs, `TeaFiMultiSend` integrates with a dedicated `PermitManager` contract that processes different types of permit signatures. Users can approve token transfers via off-chain signatures, eliminating the need for prior `approve()` transactions. The contract does not pull funds directly via allowance but instead depends on this centralized `PermitManager`, which consolidates validation logic and interacts with Uniswap's Permit2 (`0x0000000000022D473030F116dDEE9F6B43aC78BA3`).

Operator-Based Authorization

A key security feature is the use of **operator-signed EIP-712 messages** to authorize fee structures. Each batch transfer must be accompanied by a `FeeStruct` signed by an address that holds the `OPERATOR_ROLE`. These signed messages include the sender's address, token used, number of recipients, total amount, fee tokens and amounts, and a unique nonce. The contract verifies these signatures on-chain using EIP-712 hashing and

`ECDSA.recover` , ensuring that fees cannot be manipulated or spoofed. Each operator has its own per-user nonce tracking to prevent replay attacks.

Role-Based Access Control

`TeaFiMultiSend` implements `AccessControl` to manage privileges securely. The `DEFAULT_ADMIN_ROLE` is assigned to the multisig wallet and allows the holder to manage operator roles, adjust the `maxTransfers` limit, and perform emergency withdrawals. The `OPERATOR_ROLE` is assigned to off-chain services or accounts responsible for signing valid fee structures. This system ensures **granular and auditable permissions**, and role changes emit events for transparency.

Meta-Transaction Support

The contract is capable of **processing meta-transactions** through the trusted forwarder.

Architecture and Design Considerations

The `TeaFiMultiSend` contract is built for **gas efficiency**, **extensibility**, and **security**. Permit validation is outsourced to `PermitManager` , reducing code duplication and allowing easier upgrades to signature schemes. Fee validation is offloaded to trusted operators who cryptographically authorize the transaction structure. The contract also uses `ReentrancyGuard` to protect state-changing functions against reentrancy attacks. Strict checks on array lengths, token amounts, and addresses help avoid malformed inputs and runtime errors.

Roles

Admins

Administrators with the `DEFAULT_ADMIN_ROLE` can interact with the following functions:

- `function setMaxTransfers(uint256 maxTransfers_)`
- `function emergencyWithdrawErc20(address[] memory tokens)`
- `function emergencyWithdrawEth()`

Operators

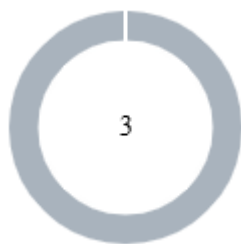
Users with the `OPERATOR_ROLE` are responsible for sign off-chain the transfers.

Users

The users can interact with the following functions:

- `function transfer(TransferStruct calldata transferData, bytes calldata tokenPermitSignature, bytes calldata permit2Signature)`
- `function batchTransfer(BatchTransferStruct calldata transferData, FeeStruct calldata fee, bytes calldata tokenPermitSignature, bytes calldata permit2Signature)`

Findings Breakdown



| | |
|-----------------------|---|
| ● Critical | 0 |
| ● Medium | 0 |
| ● Minor / Informative | 3 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|-----------------------|------------|--------------|----------|-------|
| ● Critical | 0 | 0 | 0 | 0 |
| ● Medium | 0 | 0 | 0 | 0 |
| ● Minor / Informative | 0 | 3 | 0 | 0 |

Diagnostics

● Critical ● Medium ● Minor / Informative

| Severity | Code | Description | Status |
|----------|------|---|--------------|
| ● | AAO | Accumulated Amount Overflow | Acknowledged |
| ● | CCR | Contract Centralization Risk | Acknowledged |
| ● | PTAI | Potential Transfer Amount Inconsistency | Acknowledged |

AAO - Accumulated Amount Overflow

| | |
|-------------|-------------------------|
| Criticality | Minor / Informative |
| Location | TeaFiMultiSend.sol#L169 |
| Status | Acknowledged |

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

While highly unlikely in the `batchTransfer` the entries of the `amounts` array are added to the `distributed` inside an `unchecked` block.

```
unchecked {  
    distributed += transferData.amounts[i];  
}
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

Team Update

The team has acknowledged that this is not a security issue and states: *The overflow concern is mathematically possible but economically and technically infeasible due to the use of uint256, bounded iteration under gas limits, and the internal nature of the variable. All transfer amounts are aggregated off-chain, where overflow conditions are checked and rejected before submission. The use of unchecked is justified for gas efficiency in this context and poses no practical risk.*

CCR - Contract Centralization Risk

| | |
|-------------|---------------------------------|
| Criticality | Minor / Informative |
| Location | TeaFiMultiSend.sol#L183,195,214 |
| Status | Acknowledged |

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function setMaxTransfers(uint256 maxTransfers_) external
onlyRole(DEFAULT_ADMIN_ROLE)
function emergencyWithdrawErc20(address[] memory tokens)
external onlyRole(DEFAULT_ADMIN_ROLE)
function emergencyWithdrawEth() external
onlyRole(DEFAULT_ADMIN_ROLE)
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

Team Update

The team has acknowledged that this is not a security issue and states: *The DEFAULT_ADMIN_ROLE is assigned to a Gnosis Safe multisig wallet configured with 3 owners. Any administrative action requires a minimum of 2 confirmations, therefore the risks of centralization and unilateral control are minimal.*

PTAI - Potential Transfer Amount Inconsistency

| | |
|--------------------|---|
| Criticality | Minor / Informative |
| Location | TeaFiMultiSend.sol#L106,132,166,304,312 |
| Status | Acknowledged |

Description

The `transfer()` and `transferFrom()` functions are 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 |

Specifically, in `batchTransfer` if not permitted the amount will be transferred to the contract and then to the recipient. The actual amount transferred may differ from the amount stated in the parameter.

```
function batchTransfer(
    BatchTransferStruct calldata transferData,
    FeeStruct calldata fee,
    bytes calldata tokenPermitSignature,
    bytes calldata permit2Signature
) external payable nonReentrant {
    //...
    bool fromPermitManager = _receiveNonPermitManagerPayment(
        transferData.token,
        _msgSender(),
        address(this),
        transferData.totalAmount
    );
    //...
    fromPermitManager =
    _receiveNonPermitManagerPayment(feeToken, _msgSender(),
    feeRecipient, amount);
    //...
    _sendPayment(transferData.token, recipient, amount);
    //...
}

function _receiveNonPermitManagerPayment(**args**) internal
returns (bool fromPermitManager) {
    //...
    if (IERC20(token).allowance(from, address(this)) >= amount)
    {
        IERC20(token).transferFrom(from, recipient, amount);
        return false;
    }
    //...
}

function _sendPayment(address token, address recipient, uint256
amount) internal {
    if (token == address(0)) {
        Address.sendValue(payable(recipient), amount);
        return;
    }
    IERC20(token).safeTransfer(recipient, amount);
}
```

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

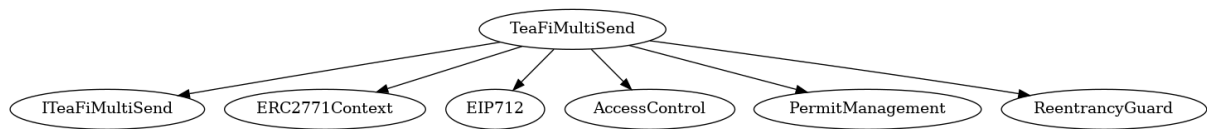
The team has acknowledged that this is not a security issue and states: *The protocol is designed to support standard ERC-20 and native tokens, which are expected to transfer the full amount specified in transfer/transferFrom/sendValue calls. Supporting tokens that apply fees or taxes during transfers would require balance-differential logic which introduces unnecessary complexity and gas overhead. Moreover, since users are free to import and transfer any token through the protocol, it's ultimately their responsibility to be aware of whether the token's behavior will affect the final amount received.*

Functions Analysis

| Contract | Type | Bases | | |
|-----------------------|---------------------------------|--|------------|---|
| | Function Name | Visibility | Mutability | Modifiers |
| | | | | |
| TeaFiMultiSend | Implementation | ITeaFiMultiSend, ERC2771Context, EIP712, AccessControl, PermitManagement, ReentrancyGuard | | |
| | | Public | ✓ | ERC2771Context EIP712 PermitManagement nonZeroUint256 |
| | transfer | External | ✓ | nonReentrant nonZeroUint256 |
| | batchTransfer | External | Payable | nonReentrant |
| | setMaxTransfers | External | ✓ | onlyRole nonZeroUint256 |
| | emergencyWithdrawErc20 | External | ✓ | onlyRole |
| | emergencyWithdrawEth | External | ✓ | onlyRole |
| | _inputGuard | Private | ✓ | nonZeroUint256 |
| | _verifySignature | Internal | ✓ | |
| | _receiveNonPermitManagerPayment | Internal | ✓ | |
| | _sendPayment | Internal | ✓ | |
| | _msgSender | Internal | | |
| | _msgData | Internal | | |

| | | | | |
|--|----------------------|----------|--|---|
| | _contextSuffixLength | Internal | | |
| | hashTypedDataV4 | External | | - |

Inheritance Graph



Summary

Tea-Fi contract implements a utility mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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

<https://www.cyberscope.io>