Cyberscope

Audit Report

Tea-Fi

May 2025

Files	TeaFiMultiSend.sol
Audited by	© cyberscope

Table of Contents

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

Source Files

Filename	SHA256
TeaFiMultiSend.sol	e92ee75aca5f57ec3f202cf7824a26c6e4f8f72b4aac38ecbd4514a1bf7e1ad0

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 (`0x000000000022D473030F116dDEE9F6B43aC78BA3`).

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

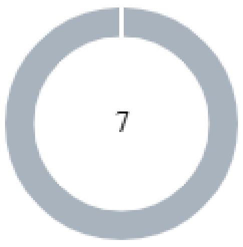
Users with the `OPERATOR_ROLE` are responsible for sign off-chain the transfers.

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	7

Severity		Unresolved	Acknowledged	Resolved	Other
🔴	Critical	0	0	0	0
🟠	Medium	0	0	0	0
⚪	Minor / Informative	7	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MEE	Missing Events Emission	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	ME	Misleading Error	Unresolved
●	AAO	Accumulated Amount Overflow	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	TeaFiMultiSend.sol#L183,194,212
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setMaxTransfers(
    uint256 _maxTransfers
) external onlyRole(DEFAULT_ADMIN_ROLE)
nonZeroUint256(_maxTransfers) {
    maxTransfers = _maxTransfers;
}
function emergencyWithdrawErc20(address[] memory tokens)
external onlyRole(DEFAULT_ADMIN_ROLE)
function emergencyWithdrawEth() external

onlyRole(DEFAULT_ADMIN_ROLE)
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	TeaFiMultiSend.sol#L183,194,212
Status	Unresolved

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.

ME - Misleading Error

Criticality	Minor / Informative
Location	TeaFiMultiSend.sol#L225
Status	Unresolved

Description

The contract has error names that do not accurately reflect the actual checks. Specifically, in the `_inputGuard` function it is checked if the length of the recipients array is different than the length of the amount array or if it is equal to zero, however the error message only covers the miss-match of the length.

```
if (len != transferData.amounts.length || len == 0) revert
ArrayLengthMismatch();
```

Recommendation

It's always a good practice for the contract to contain error messages that accurately reflect the checks that are being used for.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	TeaFiMultiSend.sol#L169
Status	Unresolved

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.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	TeaFiMultiSend.sol#L106,132,166,300,308
Status	Unresolved

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
```

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TeaFiMultiSend.sol#L184
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256 _maxTransfers
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	TeaFiMultiSend.sol#L301
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(token).transferFrom(from, recipient, amount)
```

Recommendation

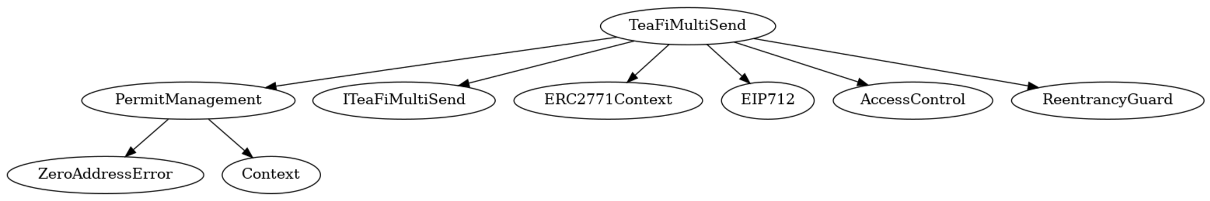
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

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

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