Cyberscope

Audit Report

Tea-Fi

May 2025

Files ProxyTrade, SynthToken, SynthTokenFactory, TeaFiRelayer, TeaFiTrustedForwarder, Authorizable, Permitable, PermitManagement, PemritManager, DecimalsCorrectionLib

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Risk Classification **3**
Review **4**
Audit Scope 4
Audit Updates 4
Source Files 5
Overview **6**
ProxyTrade Contract 6
makePublicSwap Functionalities 6
Synth Token Wrapping and Unwrapping 6
DOP Relay Support 7
TeaFiRelayer Contract 7
relayCall Functionality 7
relayCallBatch and Payment Verification 7
TeaFiTrustedForwarder Contract 8
execute Functionality 8
executeBatch Functionality 8
General Functionalities 8
Authorities Functionalities 8
SynthToken Contract 9
wrap Functionality 9
unwrap Functionality 9
General Functionalities 9
Authorities Functionalities 9
SynthTokenFactory Contract 10
createSynthTokens Functionality 10
General Functionalities 10
Findings Breakdown **11**
Diagnostics **12**
MC - Missing Check 13
Description 13
Recommendation 13
MNFC - Missing Non-Reentrance Functionality Concern 14
Description 14
Recommendation 14
CCR - Contract Centralization Risk 15
Description 15
Recommendation 18
Team Update 18

MPC - Merkle Proof Centralization 19
Description 19
Recommendation 20
Team Update 20
PCMPE - Potential Cross-Chain Merkle Proof Exploit 21
Description 21
Recommendation 21
Team Update 21
PTAI - Potential Transfer Amount Inconsistency 22
Description 22
Recommendation 23
Team Update 23
URT - Unaddressed Refunded Tokens 24
Description 24
Recommendation 24
L17 - Usage of Solidity Assembly 25
Description 25
Recommendation 26
Team Update 26
Functions Analysis **27**
Inheritance Graph **33**
Summary **34**
Disclaimer **35**
About Cyberscope **36**

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 Scope

The current contract heavily relies on the `trustedForwarder_` external contract, to perform crucial functionalities. While this dependency enables important functionality, any interactions with this external contract should be carefully reviewed and handled, as it is beyond the scope of this audit. The behavior and security of this external contract have not been assessed as part of this audit, and any interactions with it should be treated with caution to mitigate potential risks.

Audit Updates

Initial Audit	30 Sep 2024 https://github.com/cyberscope-io/audits/blob/main/1-tea/v1/forwarderSynthSwap.pdf
Corrected Phase 2	25 Oct 2024 https://github.com/cyberscope-io/audits/blob/main/1-tea/v2/forwarderSynthSwap.pdf
Corrected Phase 3	07 May 2025 https://github.com/cyberscope-io/audits/blob/main/1-tea/v3/forwarderSynthSwap.pdf
Corrected Phase 4	27 May 2025

Source Files

Filename	SHA256
TeaFiTrustedForwarder.sol	04515c581d755f3348b39b7af8f1775bd22a524afc66cc911f0aaccd9bc5af11
TeaFiRelayer.sol	cf70cd514ea6f436b971030b59e40f1c1bb5c97e44538ca8c8cf0dff4d892d2c
SynthTokenFactory.sol	c5dc484200d9cc709dd303830d7a11c53699a3d9f13d03d75a1d14cb400d5360
SynthToken.sol	6399e37191855d92a8ce16fc9a67647c64dd3f20b31a8078aa439268afb624c6
ProxyTrade.sol	817caffa38f0999c9c591d57c9dca5624d58548fe4c946f1d15ad5bdb1466f4e
PermitManager.sol	19303a4ca1b1a5348aec81f0af7ac60eec9c255d9fe803a31127949392ec45f4
PermitManagement.sol	a3d5ae129978749fdec94246e1b6735bf0a869ba4ad69a08ecda8359520258d9

Overview

ProxyTrade Contract

The `ProxyTrade` smart contract is a robust and secure proxy designed to facilitate token swaps via the 1inch DEX aggregator. It integrates several advanced mechanisms to enhance usability and security, including support for meta-transactions, synthetic tokens, and token permits. The contract supports interactions with both native and non-native tokens, with full compatibility for wrapping, unwrapping, and delegated execution via trusted contracts within the DOP ecosystem.

`makePublicSwap` Functionalities

The `makePublicSwap` function is responsible for managing both standard and synthetic token swaps. When synthetic tokens are involved, it first unwraps them to access the underlying asset. It then performs the swap via 1inch and wraps the resulting tokens if needed before sending them to the user. The contract ensures that all underlying tokens match the expected swap parameters and that appropriate token allowances are in place before executing the swap.

For native ETH swaps, the `makePublicSwapWithNative` function handles receiving ETH directly. It validates that the correct amount of ETH is provided, conducts the swap using the native asset routing via 1inch, and refunds any excess ETH to the user. This function also includes logic to convert the resulting tokens into their synthetic form if necessary.

Synth Token Wrapping and Unwrapping

To support synthetic token flows, the contract includes two dedicated methods. The `wrapNativeToSynth` function converts incoming ETH into WETH and then wraps it into a synthetic token like `tWETH`, sending the result to the specified recipient. Conversely, the `unwrapSynthToNative` function allows users to convert synthetic tokens back into the native ETH form. This is achieved by first unwrapping the synth into WETH, then withdrawing it into ETH and sending it to the user.

DOP Relay Support

The contract is also designed to operate within the DOP framework by allowing token relay functionality. The `relayToDopWithApproval` function combines token transfer approval with an authorized call to a trusted DOP contract. This function uses Permit2 or token-specific signatures to securely receive tokens from the user before executing a delegated action.

For more general-purpose execution, the `relayToDop` function allows direct calls to either the DOP relayer or smart wallet, as long as the address is pre-approved. This ensures that contract interactions are limited to a secure and predictable set of trusted contracts.

TeaFiRelayer Contract

The `TeaFiRelayer` contract is a specialized smart contract designed to securely manage meta-transactions and token-based payments within a gasless transaction framework. It supports advanced functionality for relaying calls through a trusted forwarder while integrating permit-based token approval mechanisms. The contract emphasizes strict role-based access control to maintain a secure and modular system, allowing only authorized operators and token limit managers to perform sensitive actions.

`relayCall` Functionality

The primary method for processing a meta-transaction is the `relayCall` function. This method allows an authorized operator to relay a transaction on behalf of a user. Before relaying, the contract collects token payment from the user by invoking `_receivePayment`, which validates the payment parameters and executes a transfer through the permit manager. Only after successful payment does the contract forward the request to the trusted forwarder for execution.

`relayCallBatch` and Payment Verification

To support higher throughput and operational efficiency, the `relayCallBatch` function allows authorized operators to execute multiple relay calls within a single transaction. This method iterates over a set of user requests and their corresponding payment data, processing each one sequentially. It ensures the integrity of input by validating that all input arrays are of equal length and throws an error if they are not.

A critical internal function, `_receivePayment`, handles the logic of verifying a payment's legitimacy and executing the actual token transfer. It checks the token and amount against a Merkle proof, validating that the token and its limit are part of the authorized set. If verification passes and the amount is within the allowed limit, the contract constructs a `PermitTransferParams` struct and calls `executePermitTransfer` on the permit manager to securely move tokens from the payer to the treasury.

TeaFiTrustedForwarder Contract

The `TeaFiTrustedForwarder` contract serves as a secure meta-transaction forwarder with role-based access control, allowing only authorized relayers to execute transactions. By whitelisting specific relayer contracts, it adds a layer of security to the forwarding process, ensuring that only trusted entities can interact with the contract.

`execute` Functionality

Allows authorized relayers with `PROXY_ROLE` to execute a single meta-transaction on behalf of a user. The function securely forwards the transaction request, maintaining strict access control.

`executeBatch` Functionality

Enables relayers to process multiple meta-transactions in a batch, reducing gas costs and improving transaction efficiency. Only whitelisted relayers can call this function, further securing the batch execution.

General Functionalities

The constructor assigns initial admin rights, and the `setupRoles` function is used to designate multisig wallets as admins and assign the `PROXY_ROLE` to trusted relayer addresses. Once initialized, the contract prevents further role changes, ensuring security and stability.

Authorities Functionalities

Implements `AccessControl` to manage roles like `DEFAULT_ADMIN_ROLE` and `PROXY_ROLE`. These roles are set during deployment and are locked post-initialization, allowing only trusted relayers to interact with the forwarder and ensuring robust security.

SynthToken Contract

The `SynthToken` contract is an ERC20-compatible token that represents synthetic assets tied to an underlying asset. It supports meta-transactions through a trusted forwarder and uses Permit2 for off-chain token approvals. The contract allows for wrapping and unwrapping of assets, minting synthetic tokens upon wrapping and burning them during unwrapping, with pausing capabilities for added control.

`wrap` Functionality

The `wrap` function enables users to convert underlying assets into synthetic tokens. It transfers the underlying asset to the treasury and mints an equivalent amount of synthetic tokens, supporting both standard ERC20 approvals and Permit2 for token transfers.

`unwrap` Functionality

This function allows users to convert synthetic tokens back into the underlying asset by burning the synthetic tokens and transferring the corresponding amount from the treasury to the user.

General Functionalities

The constructor sets up the synthetic token's name, symbol, underlying asset, treasury, and trusted forwarder, assigning the factory as the contract's owner. It integrates `ERC2771Context` for meta-transaction support and includes `pause` and `unpause` methods to restrict all token activities when needed.

Authorities Functionalities

The factory address manages pausing, with role-based restrictions ensuring that only the factory can invoke `pause` and `unpause`. This provides controlled access over token transfers, minting, and burning, enhancing the contract's security and flexibility.

SynthTokenFactory Contract

The `SynthTokenFactory` is a factory contract designed to create synthetic tokens by wrapping underlying assets. It allows authorized operators to define custom settings for each synthetic token, including the underlying asset and the treasury address. Additionally, it provides mechanisms for token managers to pause or unpause tokens.

`createSynthTokens` Functionality

This function enables operators to create synthetic tokens by specifying parameters such as the underlying asset and treasury address. Each token is generated with a name and symbol derived from the underlying asset's symbol (e.g., "Tea-Wrapped" + asset symbol). If no treasury address is specified, the global treasury is used.

General Functionalities

The constructor assigns roles for token managers and operators, designating the multisig wallet address as the default admin while granting additional addresses operator and token manager privileges. The factory maintains essential global settings, including the addresses for the global treasury, trusted forwarder, and permit2, which can be updated by the admin with corresponding events emitted to ensure transparency. Token managers are also authorized to pause or unpause synthetic tokens, allowing for effective management of token availability.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	3	5	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	MC	Missing Check	Unresolved
	MNFC	Missing Non-Reentrance Functionality Concern	Unresolved
	CCR	Contract Centralization Risk	Acknowledged
	MPC	Merkle Proof Centralization	Acknowledged
	PCMPE	Potential Cross-Chain Merkle Proof Exploit	Acknowledged
	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
	URT	Unaddressed Refunded Tokens	Unresolved
	L17	Usage of Solidity Assembly	Acknowledged

MC - Missing Check

Criticality	Minor / Informative
Location	ProxyTrade.sol#L96
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

In `makePublicSwap` a check is missing to ensure that `swap.desc.amount` holds a non-zero value.

```
function makePublicSwap(
    OneInchSwap calldata swap,
    SynthSupport calldata synthSupport,
    bytes calldata permitSingleSignature,
    bytes calldata tokenSignature
) external override {
    //...
    IAggregationRouterV6(oneInch).swap(swap.executor,
swap.desc, swap.data);
    //...
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MNFC - Missing Non-Reentrance Functionality Concern

Criticality	Minor / Informative
Location	ProxyTrade.sol#L96,163
Status	Unresolved

Description

There are certain functions in the contract that are not protected by potential reentrancies. `makePublicSwap` for example can transfer ETH to an account specified by the caller of the function. Since the function is not protected from reentrancy, the account could call other functions of the contract potentially disturbing the intended workflow.

```
function makePublicSwap(
    OneInchSwap calldata swap,
    SynthSupport calldata synthSupport,
    bytes calldata permitSingleSignature,
    bytes calldata tokenSignature
) external override

function wrapNativeToSynth(address recipient) external payable
override
```

Recommendation

The team could consider adding the non-reentrant modifier to all external and public functions to protect the contract from potential reentrancies.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	SynthTokenFactory.sol#L115,152,163,174 TeaFiRelayer.sol#L77,84,107 TeaFiTrustedForwarder.sol#L37,59,68 ProxyTrade.sol#L219,224
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.

Specifically, the following roles have significant authority over key contract functions:

- **OPERATOR_ROLE**: Has the authority to create new synthetic tokens with the provided settings and parameters, and execute relay calls.
- **DEFAULT_ADMIN_ROLE**: Has the authority to set the global treasury and the trusted forwarder address, and set up roles.
- **TOKEN_MANAGER**: Can pause and unpause transactions.
- **TOKEN_LIMIT_MANAGER_ROLE**: Can change token limits.
- **PROXY_ROLE**: Can execute single and multiple transactions via the relayer.

This concentration of power could lead to potential abuse or mismanagement if these roles are not properly decentralized or adequately secured.

```
function createSynthTokens(
    TokenSettings[] memory args
) external onlyRole(OPERATOR_ROLE) returns (address[] memory) {
    address[] memory tokens = new address[](args.length);
    ...
    return tokens;
}

function setGlobalTreasury(address newTreasury) external
virtual onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
    factorySettings.globalTreasury = newTreasury;

    emit GlobalTreasurySet(newTreasury);
}

function setTrustedForwarder(address newForwarder) external
virtual onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
    factorySettings.trustedForwarder = newForwarder;

    emit TrustedForwarderSet(newForwarder);
}


function pauseTokens(address[] calldata tokens) external
onlyRole(TOKEN_MANAGER) {
    ...
        SynthToken(tokens[i]).pause();
    }
}

function unpauseTokens(address[] calldata tokens) external
onlyRole(TOKEN_MANAGER) {
    ...
        SynthToken(tokens[i]).unpause();
    }
}
```

```
function changeTokenLimit(bytes32 paymentTokenLimitRoot_)
external override onlyRole(TOKEN_LIMIT_MANAGER_ROLE)

function relayCallBatch(
    ERC2771Forwarder.ForwardRequestData[] calldata requests,
    PaymentData[] calldata paymentDatas,
    bytes[] calldata tokenSignatures,
    bytes[] calldata permitSingleSignatures,
    bytes32[][] calldata merkleProofs
) external override onlyRole(OPERATOR_ROLE)

function relayCall(
    ERC2771Forwarder.ForwardRequestData calldata request,
    PaymentData calldata paymentData,
    bytes calldata tokenSignature,
    bytes calldata permitSingleSignature,
    bytes32[] calldata merkleProof
) public override onlyRole(OPERATOR_ROLE)
checkSupplierAndSigner(request.from, paymentData.payer) {
    //...
    trustedForwarder.execute(request);
}
```

```
function setupRoles(address multisigWallet, address[] memory
relayers) external onlyRole(DEFAULT_ADMIN_ROLE) {
    ...
    initialized = true;
}
function execute(ForwardRequestData calldata request) public
payable override onlyRole(PROXY_ROLE) {
    super.execute(request);
}
function executeBatch(
    ForwardRequestData[] calldata requests,
    address payable refundReceiver
) public payable override onlyRole(PROXY_ROLE) {
    super.executeBatch(requests, refundReceiver);
}
```

```
function withdrawNative() external onlyOwner
function withdrawTokens(IERC20 token) external onlyOwner
```

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 a multisig contract where the administrators are 3 people and to implement a transaction, at least 2 signatures are needed, which minimizes centralization and all other operators/managers will be reliably protected. If we suddenly lose access to one of OPERATOR_ROLE or TOKEN_MANAGER or TOKEN_LIMIT_MANAGER_ROLE, the DEFAULT_ADMIN_ROLE can revoke the role for them. For PROXY_ROLE it is impossible to do something critical because you need a user signature.

MPC - Merkle Proof Centralization

Criticality	Minor / Informative
Location	TeaFiRelayer.sol#L70,148
Status	Acknowledged

Description

The contract uses a Merkle Proof mechanism. The verification process is based on an off-chain configuration. The contract's administrators are responsible for updating the in-chain “Merkle Root” in order to validate correctly the provided message.

```
function changeTokenLimit(bytes32 paymentTokenLimitRoot_)
external override onlyRole(TOKEN_LIMIT_MANAGER_ROLE) {
    if (paymentTokenLimitRoot_ == bytes32(0)) revert
MerkleRootCannotBeZero();

    paymentTokenLimitRoot = paymentTokenLimitRoot_;
    emit TokenLimitChanged(paymentTokenLimitRoot_);
}

function _verifyPayment(address token, uint256 limit, bytes32[]
calldata merkleProof) private view returns (bool) {
    bytes32 leaf = keccak256(abi.encode(token, limit));
    return MerkleProof.verify(merkleProof,
paymentTokenLimitRoot, leaf);
}
```

Recommendation

We state that the Merkle Proof algorithm is required for proper protocol operations and gas consumption decrease. Thus, we emphasize that the Merkle proof algorithm is based on an off-chain mechanism. Any off-chain mechanism could potentially be compromised and affect the on-chain state unexpectedly. The team should carefully manage the private keys of administrator accounts. 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.

Team Update

The team has acknowledged that this is not a security issue and states: *Updating the Merkle root is a necessary part of the protocol's operation, allowing administrators with TOKEN_LIMIT_MANAGER_ROLE to adjust token limits for relayer payments as needed. This role is assigned to a multisig wallet managed by 3 signers and to confirm a transaction, at least 2 signatures are needed, which minimizes centralization risk.*

PCMPE - Potential Cross-Chain Merkle Proof Exploit

Criticality	Minor / Informative
Location	TeaFiRelayer.sol#L159
Status	Acknowledged

Description

The contract uses the `MerkleProof` to verify the payment of the `relayCall`. However during the validation, the chain Id is not accounted for. This means that the same proof could be used in multiple chains provided that the same root is also used.

```
function _verifyPayment(address token, uint256 limit, bytes32[]
calldata merkleProof) private view returns (bool) {
    bytes32 leaf = keccak256(abi.encode(token, limit));
    return MerkleProof.verify(merkleProof,
paymentTokenLimitRoot, leaf);
}
```

Recommendation

The team should consider adding the chain id in the leaf to ensure that the same proof cannot be used in different chains even if the proof remains the same.

Team Update

The team has acknowledged that this is not a security issue and states: *The concern is mitigated by the fact that token addresses and limit values are inherently chain-specific, resulting in unique Merkle leaves and paymentTokenLimitRoot on each network. If stronger isolation is required, alternative strategies for namespacing leaves may be applied.*

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	SynthToken.sol#L96
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

```
SafeERC20.safeTransferFrom(IERC20(underlyingAsset),
_msgSender(), treasury, 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:

SynthToken contracts can only be created by the Tea-Fi team, which designed the underlying assets to be free of transfer fees. The addition of some additional checks complicates the execution of the contract and increases the consumption of gas.

URT - Unaddressed Refunded Tokens

Criticality	Minor / Informative
Location	ProxyTrade.sol#L96,143
Status	Unresolved

Description

`ProxyTrade` interacts with the `oneInch` router via `makePublicSwap` and `makePublicSwapWithNative`. The users are able to use these functions to swap a certain amount of tokens. However the router may not be able to swap all tokens and the excess amount is transferred back to the contract. This amount is not handled by the contract.

```
function makePublicSwap(
    OneInchSwap calldata swap,
    SynthSupport calldata synthSupport,
    bytes calldata permitSingleSignature,
    bytes calldata tokenSignature
) external override {
    //...
    IAggregationRouterV6(oneInch).swap(swap.executor,
swap.desc, swap.data);
    //...
}

function makePublicSwapWithNative(
    OneInchSwap calldata swap,
    SynthSupport calldata synthSupport
) external payable override nonReentrant {
    IAggregationRouterV6(oneInch).swap{value:
swap.desc.amount}(swap.executor, swap.desc, swap.data);
}
```

Recommendation

It is recommended that the contract handles the refunded amount from the swap with the router. This will ensure that the contract does not keep excess tokens.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	components/Permitable.sol#L103
Status	Acknowledged

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly ("memory-safe") {
            // solhint-disable-line no-inline-assembly
            let ptr := mload(0x40)

            // Switch case for different permit lengths,
indicating different permit standards
            switch permit.length
...
            }
            // Unknown
            default {
                mstore(ptr, permitLengthError)
                revert(ptr, 4)
            }
        }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Team Update

The team has acknowledged that this is not a security issue and states: *The private _tryPermit function, sourced from 1inch's AggregationRouterV6 (0x111111125421cA6dc452d289314280a0f8842A65), uses assembly for efficient and secure permit validation.*

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TeaToken	Implementation	ERC20, ERC2771Context, ERC20Permit, ERC20Votes, ZeroAddressError, Ownable, ERC20Burnable		
		Public	✓	ERC20 ERC2771Context ERC20Permit Ownable
	mint	External	✓	onlyOwner
	clock	Public		-
	CLOCK_MODE	Public		-
	nonces	Public		-
	_update	Internal	✓	
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
	hashTypedDataV4	External		-
TeaFiTrustedForwarder	Implementation	ERC2771Forwarder, AccessControl, ZeroAddressError		

		Public	✓	ERC2771Forwarder
	setupRoles	External	✓	onlyRole
	execute	Public	Payable	onlyRole
	executeBatch	Public	Payable	onlyRole
	hashTypedDataV4	External		-
TeaFiRelayer	Implementation	Authorizable, ITeaFiRelayer		
		Public	✓	Authorizable
	changeTokenLimit	External	✓	onlyRole
	relayCallBatch	External	✓	onlyRole
	relayCall	Public	✓	onlyRole checkSupplierAndSigner
	_receivePayment	Private	✓	
	_verifyPayment	Private		
SynthTokenFactory	Implementation	AccessControl		
		Public	✓	-
	createSynthTokens	External	✓	onlyRole
	setGlobalTreasury	External	✓	onlyRole
	setTrustedForwarder	External	✓	onlyRole
	pauseTokens	External	✓	onlyRole
	unpauseTokens	External	✓	onlyRole

SynthToken	Implementation	ERC20, ISynthToken, ERC2771Context, ERC20Permit, Permitable, ERC20Pausable		
		Public	✓	ERC20 ERC2771Context Permitable ERC20Permit
	wrap	External	✓	-
	wrap	Public	✓	-
	wrap	Public	✓	checkZeroAmount
	unwrap	External	✓	checkZeroAmount
	pause	External	✓	onlyFactory
	unpause	External	✓	onlyFactory
	_update	Internal	✓	
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
	hashTypedDataV4	External		-
ProxyTrade	Implementation	ERC2771Context, Ownable, IProxyTrade, PermitManagement, ReentrancyGuard		
		Public	✓	ERC2771Context Ownable PermitManagement
		External	Payable	-

	makePublicSwap	External	✓	-
	makePublicSwapWithNative	External	Payable	nonReentrant
	wrapNativeToSynth	External	Payable	-
	unwrapSynthToNative	External	✓	nonReentrant
	relayToDopWithApproval	External	✓	-
	relayToDop	Public	✓	nonReentrant
	withdrawNative	External	✓	onlyOwner
	withdrawTokens	External	✓	onlyOwner
	_afterOneInchSwap	Internal	✓	
	_checkAllowance	Internal	✓	
	_sendNativeAsset	Private	✓	
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
PermitManager	Implementation	Permitable, IPermitManager, AccessControl, EIP712		
		Public	✓	Permitable EIP712
	executePermitTransferBatch	External	✓	onlyRole
	executePermitTransfer	Public	✓	onlyRole
	addSpenders	External	✓	onlyRole
	removeSpenders	External	✓	onlyRole

PermitManagement	Implementation	ZeroAddressError, Context		
		Public	✓	-
	_receivePayment	Internal	✓	
Permitable	Implementation	ZeroAddressError, ITokenPermitSignatureDetails, Context		
		Public	✓	-
	_makeTokenPermit	Internal	✓	
	_makePermit2	Internal	✓	
	_receivePayment	Internal	✓	
Permitable	Implementation	ZeroAddressError		
		Public	✓	-
	_makeTokenPermit	Internal	✓	
	_makePermit2	Internal	✓	
	_transferPayment	Internal	✓	
	_safePermit	Private	✓	
	_tryPermit	Private	✓	
DecimalsCorrectionLib	Library			
	decimalsCorrection	Internal		

Authorizable	Implementation	AccessControl, ZeroAddressError		
		Public	✓	-
	_setupRoles	Internal	✓	

Inheritance Graph

For the detailed flow inheritance graph image, please refer to the link provided below:

inheritance_graph_forwarderSynthSwap.png

Summary

The Tea-Fi suite of contracts implements a comprehensive system for facilitating token swaps, synthetic asset creation, meta-transactions, and role-based access control. This audit investigates security vulnerabilities, business logic concerns, and potential improvements in the use of trusted forwarders, the Permit2 system, and role-based governance across the contracts.

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