



Cyberscope

Audit Report

Tea-Fi

May 2025

Files ProxyTrade, SynthToken, SynthTokenFactory, TeaFiRelayer,
TeaFiTrustedForwarder, Authorizable, Permittable, PermitManagement,
PermitManager, 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
CAM - Contract Allowance Manipulation	13
Description	13
Recommendation	14
IEVC - Insufficient EOA Validation Check	15
Description	15
Recommendation	15
MC - Missing Check	16
Description	16
Recommendation	16
PCMPE - Potential Cross-Chain Merkle Proof Exploit	17

Description	17
Recommendation	17
AME - Address Manipulation Exploit	18
Description	18
Recommendation	19
CCR - Contract Centralization Risk	20
Description	20
Recommendation	23
Team Update	23
MPC - Merkle Proof Centralization	24
Description	24
Recommendation	25
PTAI - Potential Transfer Amount Inconsistency	26
Description	26
Recommendation	27
Team Update	27
L04 - Conformance to Solidity Naming Conventions	28
Description	28
Recommendation	28
L17 - Usage of Solidity Assembly	29
Description	29
Recommendation	29
Team Update	30
Functions Analysis	31
Inheritance Graph	36
Summary	37
Disclaimer	38
About Cyberscope	39

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

Source Files

Filename	SHA256
TeaFiTrustedForwarder.sol	0aa017a336a9e53759744cdc9e0ea17e4b6be7bd20dc6d7aad7656745aef9766
TeaFiRelayer.sol	37201375cf12703fe9aae53a80c4d320bc0c1b2c43454a4954bd0f86ea95f765
SynthTokenFactory.sol	9b7c0b2aec72bcbf7e182f91d17ed41ef4a94474afc1f444d7103916bd79cf7a
SynthToken.sol	6411648bb5994c1aa4eb18dd8397532004102b79d553af5ac4cf0ef8e6c0c3ec
ProxyTrade.sol	4d53b1dfb78da0c77f9a56255efb0b9d5dab08f5b67b7b69c97d48c4b88f284d
PermitTable.sol	d1e17740aa7091cefca5824fc8936cbd25651dfd8cd0bc892f96fce6737cc663
PermitManager.sol	19303a4ca1b1a5348aec81f0af7ac60eec9c255d9fe803a31127949392ec45f4
PermitManagement.sol	dcf0d340ca93b3a2fda713d78c64194137a58b5ce3e4f83546b9a5d101d08e6a
DecimalsCorrectionLib.sol	366fddecaad707407227e2528418ecf3615bc4c503c8d6236fca66b6f7031010
Authorizable.sol	191beadf6c7cf4cbad63590072285ec4aa7ca7b7591947730a9de1ec555339e

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.

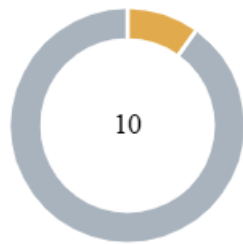
`createSynthTokens` Functionality

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	1
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	6	3	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	CAM	Contract Allowance Manipulation	Unresolved
●	IEVC	Insufficient EOA Validation Check	Unresolved
●	MC	Missing Check	Unresolved
●	PCMPE	Potential Cross-Chain Merkle Proof Exploit	Unresolved
●	AME	Address Manipulation Exploit	Unresolved
●	CCR	Contract Centralization Risk	Acknowledged
●	MPC	Merkle Proof Centralization	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Acknowledged
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L17	Usage of Solidity Assembly	Acknowledged

CAM - Contract Allowance Manipulation

Criticality	Medium
Location	ProxyTrade.sol#L121,152,236,253
Status	Unresolved

Description

`makePublicSwap` and `makePublicSwapWithNative` allow users to use any arguments for the `swap` and `synthSupport`. This allows them to potentially add malicious contract addresses as inputs to the function. In case the `synthSupport.destToken` is a malicious contract address it could be enabled to receive max allowance for the `swapDesc.dstToken`. This may eventually allow the user to receive the entire balance of the token from the contract.

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

function makePublicSwapWithNative(OneInchSwap calldata swap,
SynthSupport calldata synthSupport) external payable override
nonReentrant {
    //...
    _afterOneInchSwap(swap.desc, synthSupport);
    //...
}

function _afterOneInchSwap(IAggregationRouterV6.SwapDescription
calldata swapDesc, SynthSupport calldata synthSupport) internal
{
    //...
    _checkAllowance(swapDesc.dstToken, synthSupport.destToken,
balance, true);
    //...
}

function _checkAllowance(IERC20 token, address receiver,
uint256 amount, bool maxApproval) internal {
    if (token.allowance(address(this), receiver) < amount) {
        if (maxApproval) {
            token.safeIncreaseAllowance(receiver,
type(uint256).max);
        } else {
            token.safeIncreaseAllowance(receiver, amount);
        }
    }
}
```

Recommendation

The contract should enable checks to ensure that each argument added by users is legit and safe to call. Additionally the team should only provide the allowance necessary during `_checkAllowance`. This will increase the contracts protection against other malicious contracts.

IEVC - Insufficient EOA Validation Check

Criticality	Minor / Informative
Location	ProxyTrade.sol#L131
Status	Unresolved

Description

ProxyTrade's `makePublicSwapWithNative` function checks if the `msg.sender` is also the `tx.origin` and if not the function reverts. This check is used to only allow externally owned accounts to use this function. However due to the changes introduced to EIP-7702 this check is not sufficient as externally owned accounts can now execute smart contract code.

```
if (tx.origin != _msgSender()) revert OnlyEOA();
```

Recommendation

It is recommended that the team takes into consideration the changes introduced to the [EIP-7702](#) and make the necessary changes to ensure the validity of the code.

MC - Missing Check

Criticality	Minor / Informative
Location	ProxyTrade.sol#L127
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.

Specifically, in `makePublicSwapWithNative` a check is missing to ensure that `swap.desc.amount` is greater than zero.

```
function makePublicSwapWithNative (OneInchSwap calldata swap,
SynthSupport calldata synthSupport
) external payable override nonReentrant {
    if (tx.origin != _msgSender()) revert OnlyEOA();
    if (address(swap.desc.srcToken) != nativeOneInch) revert
OnlyNativeAsset();
    if (msg.value < swap.desc.amount) revert
NotEnoughNativeAsset();
    //...
}
```

Recommendation

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

PCMPE - Potential Cross-Chain Merkle Proof Exploit

Criticality	Minor / Informative
Location	TeaFiRelayer.sol#L148
Status	Unresolved

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.

AME - Address Manipulation Exploit

Criticality	Minor / Informative
Location	ProxyTrade.sol#L84,127,217
Status	Unresolved

Description

The contract's design includes functions that accept external contract addresses as parameters without performing adequate validation or authenticity checks. This lack of verification introduces a significant security risk, as input addresses could be controlled by attackers and point to malicious contracts. Such vulnerabilities could enable attackers to exploit these functions, potentially leading to unauthorized actions or the execution of malicious code under the guise of legitimate operations.

```
function makePublicSwap(OneInchSwap calldata swap, SynthSupport
calldata synthSupport, bytes calldata permitSingleSignature,
bytes calldata tokenSignature) external override {
    //...
    address underlyingAsset =
    ISynthToken(synthSupport.srcToken).underlyingAsset();
    //...
    ISynthToken(synthSupport.srcToken).unwrap(amount,
address(this));
    //...
}

function makePublicSwapWithNative(OneInchSwap calldata swap,
SynthSupport calldata synthSupport) external payable override
nonReentrant {
    //...
    _afterOneInchSwap(swap.desc, synthSupport);
    //...
}

function _afterOneInchSwap(
    IAggregationRouterV6.SwapDescription calldata swapDesc,
    SynthSupport calldata synthSupport
) internal {
    //...
    address underlyingAsset =
    ISynthToken(synthSupport.destToken).underlyingAsset();
    //...
    uint256 balance =
    swapDesc.dstToken.balanceOf(address(this));
    //...
    ISynthToken(synthSupport.destToken).wrap(balance,
_msgSender());
    //...
}
```

Recommendation

To mitigate this risk and enhance the contract's security posture, it is imperative to incorporate comprehensive validation mechanisms for any external contract addresses passed as parameters to functions. This could include checks against a whitelist of approved addresses, verification that the address implements a specific contract interface or other methods that confirm the legitimacy and integrity of the external contract. Implementing such validations helps prevent malicious exploits and ensures that only trusted contracts can interact with sensitive functions. An example of such an exploit is also described in the `CAM` section.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	SynthTokenFactory.sol#L115,152,163,174 TeaFiRelayer.sol#L57,75,86 TeaFiTrustedForwarder.sol#L35,57,66 ProxyTrade.sol#L203,208
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(address 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	Unresolved

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.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	SynthToken.sol#L95
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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	TeaFiRelayer.sol#L70 ProxyTrade.sol#L36
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.

```
bytes32 _paymentTokenLimitRoot
IWETH public immutable WETH
```

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>.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	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` (`0x11111125421cA6dc452d289314280a0f8842A65`), uses assembly for efficient and secure permit validation.*

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
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, ERC20Pauseable		
		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		
Permitable	Implementation	ZeroAddress Error		
		Public	✓	-

	_makeTokenPermit	Internal	✓	
	_makePermit2	Internal	✓	
	_transferPayment	Internal	✓	
	_safePermit	Private	✓	
	_tryPermit	Private	✓	
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	✓	
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