Cyberscope

Audit Report

Tea-Fi

April 2025

Files ProxySynthStaking.sol, ProxyStorage.sol, ProxyStakingService.sol, ProxyRewardsService.sol, ProxyEIP712Service.sol

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	10 Apr 2025

Source Files

Filename	SHA256
ProxySynthStaking.sol	0f3c2573c69e064aef24c3e133a56940145c6d10789eef8481a22fa9be27ddfe
ProxyStorage.sol	e1dd3bc573edc896aba1e8e6634a42ac2eeda90b25139260026e5a06c1929a49
ProxyStakingService.sol	0e547f3e5a9a00537f2c8c12cc6c4afe008b8a0300a78fcd6a2f4cd66149af7a
ProxyRewardsService.sol	fcc893f797efe377276931a9054126ed55d289be7530a0c16fe4c49335776a32
ProxyEIP712Service.sol	764820b4eaa0d5c851173f68676a7896cd441dce547a3a4610b073d655776559
DecimalsCorrectionLib.sol	a869119666eb6599d4fbe68f1198e3be4576dae485d97b5aa4670fea2494c956

Overview

Proxy SynthStaking Contract

The `ProxySynthStaking` contract implements proxy functionality for synthetic staking operations. Its role is to delegate and manage staking-related logic while abstracting away the underlying storage layers. The contract facilitates staking operations by allowing users to deposit tokens into a synthetic staking mechanism. The proxy layer forwards calls to the current staking implementation. It tracks staked balances and updates users' positions based on their deposits and withdrawals. In doing so, it interacts with the underlying `ProxyStorage` to maintain a persistent state.

Proxy EIP712 Service Contract

The `ProxyEIP712Service` contract serves as the utility layer handling typed data signing and message verification. This is critical for ensuring secure off-chain data integrity and structured data validation. It implements standards for hashing and signing structured data, which allows off-chain signatures to be securely verified on-chain. It validates signatures, ensuring that only authorized messages are acted upon in subsequent proxy interactions such as staking or rewards operations.

Proxy Rewards Service Contract

The `ProxyRewardsService` contract is geared toward managing rewards distribution within the staking ecosystem. Its proxy design allows the rewards calculation and distribution logic to be updated independently of user data. Once rewards are determined, the contract facilitates the collect and withdraw processes.

Proxy Staking Service Contract

The `ProxyStakingService` contract forms the core interface for staking operations within the ecosystem. It acts as a mediator between users and the underlying staking logic and storage. It provides functions for staking tokens, unstaking, and withdrawing. The contract ensures that users' actions are correctly mapped to changes in their staking positions. Uses the proxy pattern to delegate logic to the current implementation of staking

functionality. It delegates persistent state management to the `ProxyStorage` contract. This decoupling helps keep the logic modular and simplifies maintenance.

Proxy Storage Contract

The `ProxyStorage` contract acts as the persistent data repository for the proxy ecosystem, storing critical parameters across the staking rewards. It maintains storage of staking parameters, and configuration settings. This centralized store ensures data integrity across various proxy implementations. Because all contracts interact with a single storage contract, logic upgrades across multiple proxy contracts do not lead to data inconsistency. It also provides secure methods to read from and write to storage. Only authorized proxy contracts can modify the state, thereby preventing unauthorized alterations.

Roles

Admins

Administrators or the owner of the contract can interact with the following functions:

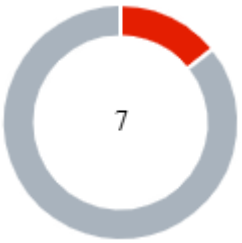
- function emergencyWithdrawErc20(address[] memory tokens,address recipient)
- function emergencyWithdrawEth(address recipient)
- function setTeaToken(address teaToken_)
- function setPermitManager(address permitManager_)
- function syncPools(uint64[] calldata poolIds)

Users

Stakers or participants can interact with:

- function stake(uint64 poolId, uint256 amount, bytes calldata permitSingleSignature, bytes calldata tokenSignature)
- function unstake(uint64 stakeId, uint256 amount)
- function withdrawStake(uint64 unstakeId)
- function collectReward(ISynthStaking.CollectRewardParam calldata param,ProxyCollectRewardParam calldata proxyParam)
- function withdrawReward(uint64 rewardId)

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	6

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	UAI	Unstake Amount Inconsistency	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	MC	Missing Check	Unresolved
	PMVR	Potential Mapping Value Reset	Unresolved
	PRE	Potential Reentrance Exploit	Unresolved
	TCV	Transfers Contract's Value	Unresolved
	L16	Validate Variable Setters	Unresolved

UAI - Unstake Amount Inconsistency

Criticality	Critical
Location	ProxyStakingService.sol#L64
Status	Unresolved

Description

The `unstake` function allows the user to specify an amount to unstake from a specific `stakeId`. The `activeStakesCount` for the staker is updated as the entire staked amount is unstaked. However, since the user can specify the amount they could choose to unstake a smaller amount meaning that `activeStakesCount` should not be updated.

```
function unstake(uint64 stakeId, uint256 amount) external
nonZeroUint256(amount) {
    //...
    implementation.unstake(stakeId, synthAmount);
    //...
    unchecked {
        --activeStakesCount[staker];
    }
    //...
}
```

Recommendation

The team is advised to check if the entire amount has been unstaked and then update `activeStakesCount` accordingly.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	ProxySynthStaking.sol#L63,84 ProxyStorage.sol#L126,136,149
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 emergencyWithdrawErc20(address[] memory tokens,address
recipient) external validAddress(recipient)
onlyRole(DEFAULT_ADMIN_ROLE)
function emergencyWithdrawEth(address recipient) external
validAddress(recipient) onlyRole(DEFAULT_ADMIN_ROLE)
function setTeaToken(address teaToken_) external
onlyRole(DEFAULT_ADMIN_ROLE) validAddress(teaToken_)
function setPermitManager(address permitManager_) external
onlyRole(DEFAULT_ADMIN_ROLE) validAddress(permitManager_)
function syncPools(uint64[] calldata poolIds) 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.

MC - Missing Check

Criticality	Minor / Informative
Location	ProxySynthStaking.sol#L63 ProxyStorage.sol#L164
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 `emergencyWithdrawErc20` a check is missing to ensure that `tokens` are not the address zero.

```
function emergencyWithdrawErc20(
    address[] memory tokens,
    address recipient
) external validAddress(recipient) onlyRole(DEFAULT_ADMIN_ROLE)
{
    uint256 length = tokens.length;
    for (uint256 i = 0; i < length; ++i) {
        address token = tokens[i];
        //...
    }
    //...
}
```

In `syncPool` a check is missing to ensure that `synthToken` and `underlyingAsset` are not addresses zero.

```
function _syncPool(uint64 poolId) private {
    ISynthStaking.PoolInfoView memory pool =
implementation.getPoolInfo(poolId);
    address synthToken = address(pool.synthToken);
    address underlyingAsset =
ISynthToken(synthToken).underlyingAsset();
    //...
}
```

Recommendation

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

PMVR - Potential Mapping Value Reset

Criticality	Minor / Informative
Location	ProxyStakingService.sol#L60,127
Status	Unresolved

Description

The contract is using the `stakeCounter` and `unstakeCounter` of `implementation` to define the `stakeId` and `unstakeId`. These counters are updated inside an `unchecked` block. While highly unlikely, if the counters reset then `ProxyStakingService` may overwrite existing `stakeUserMap` and `unstakeUserMap` values.

```
function unstake(uint64 stakeId, uint256 amount) external
nonZeroUint256(amount) {
    //...
    implementation.unstake(stakeId, synthAmount);
    uint64 unstakeId = implementation.unstakeCounter() - 1;
    unstakeUserMap[unstakeId] = ProxyUnstake(staker, stakeId,
amount);
    //...
}

function _stake(address underlyingAsset,address
synthToken,uint64 poolId,uint256 amount,bool isEthStake)
internal {
    //...
    implementation.stake(poolId, synthAmount);
    uint64 stakeId = implementation.stakeCounter() - 1;
    address staker = _msgSender();
    stakeUserMap[stakeId] = ProxyStake(staker, poolId,
isEthStake);
    //...
}
```

Recommendation

IT is recommended to check if a value of `stakeUserMap` and `unstakeUserMap` with the new key exists before updating the state with the new value.

PRE - Potential Reentrance Exploit

Criticality	Minor / Informative
Location	ProxyStakingService.sol#L71
Status	Unresolved

Description

`withdrawStake` can transfer funds to the staker. This leaves the contract in potential re-entrancy risks. The re-entrance exploit could be used by a malicious user to drain the contract's funds or to perform unauthorized actions.

```
function withdrawStake(uint64 unstakeId) external {
    address staker = _msgSender();
    //...
    bool isEthStake = stakeUserMap[stakeId].isEthStake;
    if (underlyingAsset == address(weth) && isEthStake) {
        _unwrapEth(synthToken, synthAmount, staker);
    } else {
        _unwrap(synthToken, synthAmount, staker);
    }
    //...
}
```

Recommendation

The team is advised to prevent the potential re-entrance exploit as part of the solidity best practices. Some suggestions are:

- Add lockers/mutexes in the method scope. It is important to note that mutexes do not prevent cross-function reentrancy attacks.
- Do Not allow contract addresses to receive funds.
- Proceed with the external call as the last statement of the method, so that the state will have been updated properly during the re-entrance phase.

TCV - Transfers Contract's Value

Criticality	Minor / Informative
Location	ProxySynthStaking.sol#L63,84
Status	Unresolved

Description

The contract admin has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `emergencyWithdrawErc20` to claim the tokens and `emergencyWithdrawEth` to claim all the ether.

```
function emergencyWithdrawErc20(
    address[] memory tokens,
    address recipient
) external validAddress(recipient) onlyRole(DEFAULT_ADMIN_ROLE)
{
    uint256 length = tokens.length;
    for (uint256 i = 0; i < length; ++i) {
        address token = tokens[i];
        (, bytes memory queriedBalance) = token.staticcall(
            abi.encodeWithSelector(IERC20.balanceOf.selector,
address(this))
        );
        uint256 withdrawable = abi.decode(queriedBalance,
(uint256));
        IERC20(token).safeTransfer(recipient, withdrawable);
    }
    emit EmergencyWithdrawErc20(recipient, tokens);
}

function emergencyWithdrawEth(address recipient) external
validAddress(recipient) onlyRole(DEFAULT_ADMIN_ROLE) {
    uint256 currentBalance = address(this).balance;
    if (currentBalance > 0) {
        _sendEth(currentBalance, recipient);
        emit EmergencyWithdrawEth(recipient, currentBalance);
    }
}
```

Recommendation

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

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ProxySynthStaking.sol#L70
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
(, bytes memory queriedBalance) =
token.staticcall(abi.encodeWithSelector(IERC20.balanceOf.select
or, address(this)))
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

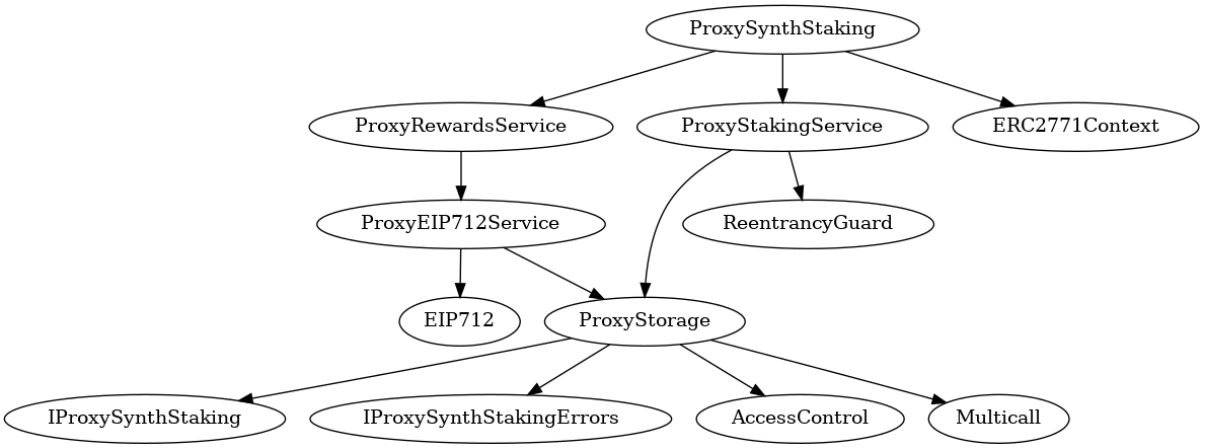
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ProxySynthStaking	Implementation	ProxyStakingService, ProxyRewardsService, ERC2771Context		
		Public	✓	validAddress ProxyStorage ERC2771Context EIP712
	emergencyWithdrawErc20	External	✓	validAddress onlyRole
	emergencyWithdrawEth	External	✓	validAddress onlyRole
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
ProxyStorage	Implementation	IProxySynthStaking, IProxySynthStakingErrors, AccessControl, Multicall		
		Public	✓	validAddress validAddress validAddress validAddress validAddress validAddress
	setTeaToken	External	✓	onlyRole validAddress
	setPermitManager	External	✓	onlyRole validAddress

	syncPools	External	✓	onlyRole
	_proxyPoolExists	Internal		
	_getProxyPoolTokens	Internal		
	_syncPool	Private	✓	
ProxyStakingService	Implementation	ProxyStorage, ReentrancyGuard		
		External	Payable	-
	stake	External	Payable	nonZeroUint256 nonReentrant
	unstake	External	✓	nonZeroUint256
	withdrawStake	External	✓	-
	_unwrapEth	Internal	✓	
	_unwrap	Internal	✓	
	_sendEth	Internal	✓	
	_stake	Internal	✓	
	_wrap	Internal	✓	
	_receiveEth	Private	✓	
	_receivePayment	Private	✓	
	_validatePool	Private		
ProxyRewardsService	Implementation	ProxyEIP712Service		
	collectReward	External	✓	-
	withdrawReward	External	✓	-
	_validateProxySignature	Private	✓	validAddress

ProxyEIP712Service	Implementation	ProxyStorage, EIP712		
	_verifyProxySignature	Internal	✓	
	_verifySignature	Internal	✓	
	hashTypedDataV4	External		-
DecimalsCorrectionLib	Library			
	decimalsCorrection	Internal		

Inheritance Graph



Summary

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

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

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