verichains

SECURITY AUDIT OF

COLEND SUBSCRIPTION TOKEN AND YIELD BOOSTED CONTRACTS

Private Report

Apr 21, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Apr 21, 2025. We would like to thank the Colend for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Colend Subscription Token and Yield Boosted Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some issues in the source code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Colend Subscription Token and Yield Boosted Contracts

The Colend Subscription Token enables users to subscribe to a specific address, allowing them to participate in the Colend Yield-Boosted Contract.

The Yield-Boosted Contract serves as a portal that supports users in supplying their balance on behalf of the address they subscribed to in the Subscription Token Contract, subsequently channeling the balance to the target Pool.

1.2. Audit Scope

This audit focused on identifying security flaws in the code and design of the Colend Subscription Token and Yield Boosted Contracts.

It was conducted on commits `ab1d57367da3e4ccb7df66f77c8b74c6a3fd082b` from the repository *https://github.com/Colend-Protocol/colend-subscription-token* and `2b52c42fb25797feec2019230eb778d339954b3b` from the repository *https://github.com/tobyColend/colend-yield-boosted-contracts*.

The latest versions of the following files were reviewed from the `colend-subscription-token` repository (commit `f4f341cd93b34a5de821f3f9a518cbd2cbe82eb4`):

SHA256 Sum	File
`5f921d23ce551c9a9d942f7675a27c9ea30cb04f5622658b74c1b7c05cd9e81a`	`src/SoulBoundToken.sol`

The latest versions of the following files were reviewed from the `colend-yield-boosted-contracts` repository (commit `2b52c42fb25797feec2019230eb778d339954b3b`):

SHA256 Sum	File
`495fb021a4a93fdf92e2796a74b6eee3ea8f159c27ada613581270a2408fdb49`	`src/SubscriptionDataIncentiveProvider.sol`
`ca564a9d1bb371d0c42c9b31422fe0491ddc49e263b38221f6f317b859cdfc23`	`src/SubscriptionPool.sol`
`3af53672836716d663c86b225b3428085913c34079e9b1a60ac0c2bebfca1870`	`src/tokens/VirtualRewardTrackingToken.sol`
`2bff6304190fa7150af953a2ee59931fa956ee7d0a9e24485e6b52ae7e5d5309`	`src/extensions/RoleManager.sol`

e11446a85b091ad1023618524041cd2d2c10aa694479c937ca9f5b9f7f0a8a32	src/interfaces/IPool.sol
d142a619caaec90c89ba761fefd347256c2975753b656f37a4f9a4325405a295	src/interfaces/ISoulboundToken.sol
ea4806fa65a62effe6387e6c60daa58539f3d00fa0f9af78a841fa18a1eb8a04	src/interfaces/ISubscriptionDataIncentiveProvider.sol
497c45037eb01558c48d2f7bce0336aea5c53082fb94957a08a66b6d4defad9c	src/interfaces/ISubscriptionPool.sol
f0081dd3c510b920b143110d4124b51d8890079987e7aca9abf7892e3354afca	src/interfaces/IVirtualRewardTrackingToken.sol

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Colend acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Colend understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Colend agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Colend will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Colend, the final report will be considered fully accepted by the Colend without the signature.

This report contains sensitive information or information that's not meant to be for the general public. These will be censored out as requested by the Colend and will be displayed as [redacted].

2. AUDIT RESULT

2.1. Overview

The Colend Subscription Token and Yield Boosted Contracts was written in `Solidity` language, with the version `^0.8.20`.

2.1.1. Soulbound Token

The `SoulBoundToken` contract is a subscription system that allows users to subscribe to whitelisted assets using `CLND` tokens with a specific `onBehalfOf` address. The `onBehalfOf` address owns the subscription tokens and is whitelisted in the `Colend-Yield-Boosted` contract. The contract also supports subscription management functions, including subscription creation, extension, auto-renewal toggling, and validity verification. For each asset type, the admin can set a token cost for the subscription. The contract defines five roles:

- `DEFAULT_ADMIN_ROLE`: The master role that can manage all other roles.
- `SUBSCRIPTION_MANAGER_ROLE`: This role supports users in auto-renewing subscriptions when they toggle the auto-renewal flag within the extension window. Managers with this role can also remove a tokenID to stop a subscription and update the tokenURI of a subscription token.
- `CONFIGURATION_MANAGER_ROLE`: Manages the contract's configuration, including token costs, subscription duration, extension windows, and asset control.
- `ACCESS_MANAGER_ROLE`: Manages addresses that can be added to the `onBehalfOf` address list.
- `TREASURY_MANAGER_ROLE`: Oversees subscriptions and related operations.

2.1.2. Colend-Yield-Boosted

This project implements a yield-boosting protocol built on top of ProtocolPool , designed to provide additional incentives to users participating in the Pool lending platform. The system tracks user deposits and distributes additional rewards through a subscription-based model.

2.1.2.1. Subscription Pool

The core contract enables users to supply assets on behalf of a designated address specified in the `SoulBoundToken` contract. These assets are then deposited into the Pool on behalf of the `onBehalfOf` address. Key features include:

- Supporting only assets and `onBehalfOf` addresses permitted by the `SoulBoundToken` contract.
- Facilitating asset deposits to the Pool on behalf of the specified address.

When assets are supplied, the contract also creates a `VirtualRewardTrackingToken` for the `onBehalfOf` address to track the rewards earned from the `SubscriptionPool` contract.

2.1.2.2. VirtualRewardTrackingToken

A specialized `ERC20` token that tracks the rewards earned by users from the `SubscriptionPool` contract:

- Represents a user's position in Pool without allowing transfers.
- Overrides `transfer`, `transferFrom`, and `approve` functions to prevent token movement.
- Integrates with Pool's incentives controller for reward distribution.
- Tracks balances adjusted by the `SubscriptionPool` contract.

2.1.2.3. SubscriptionDataIncentiveProvider

A contract used to calculate the reward that user can earn from the pool defined in `SubscriptionPool` contract.

2.2. Findings

During the audit process, the audit team had identified some issues in the source code.

#	Title	Severity	Status
1	Unauthorized Subscription Extension	HIGH	FIXED
2	Missing `withdraw` Implementation in `SubscriptionPool` Contract	LOW	ACKNOWLEDGED
3	DOS in `subscribe` logic in `SoulboundToken` contract	LOW	ACKNOWLEDGED
4	`token.caller` can be overrided in `SouldboundToken` contract	LOW	OPEN
5	Recommendation for handling unused parameters in `VirtualRewardTrackingToken` contract	INFORMATIVE	ACKNOWLEDGED
6	Unsafe Transfer Token	INFORMATIVE	OPEN

2.2.1. Unauthorized subscription extension HIGH

Affected:

- src/SouldboundToken.sol#extendMySubscription()

The extendMySubscription function allows any address to extend another user's subscription at the original payer's expense, creating a forced payment attack vector.

The function takes payment from the token's original caller (token.caller), not from the current transaction sender (msg.sender). If the original caller has approved the contract to spend their CLND tokens (which they would have done during initial subscription), any address can call this function and force them to pay for subscription extensions without their consent.

```
function extendMySubscription(address asset, address onBehalfOf) external
notBlacklisted(onBehalfOf) tokenExists(onBehalfOf, asset) nonReentrant {
    Token storage token = userTokens[onBehalfOf][asset];

    require(
        clndToken.transferFrom(token.caller, address(this), tokenCost),
        "CLND transfer failed"
    );

    // Combine multiple operations to reduce gas
    uint256 expiry = token.expiryTimestamp;
    uint256 current = block.timestamp;

    // Calculate new expiry (max of current expiry and now, plus duration)
    token.expiryTimestamp = (expiry > current ? expiry : current) + subscriptionDuration;
    token.paidMonths += 1;

    emit SubscriptionExtended(
        onBehalfOf,
        token.expiryTimestamp,
        token.paidMonths
    );
}
```

RECOMMENDATION

Implement proper access control to ensure only the token.caller, owner, or authorized addresses can extend the subscription.

UPDATES

- *Apr 21, 2025*: The issue has been acknowledged and fixed by the Colend team.

2.2.2. Missing withdraw implementation in SubscriptionPool contract LOW

Affected:

- src/SubscriptionPool.sol

The `SubscriptionPool` contract does not implement the `withdraw` function, which is described in the documentation (`subscription_pool_flows.md`). This function is essential for enabling users to withdraw their assets from the pool.

```
# 2. Withdraw flow

1. User approve `@AToken` for `@SubscriptionPool` .
2. User call to `@SubscriptionPool.withdraw()` to withdraw their asset.
3. `@SubscriptionPool` receives user's `@AToken` and call to `@Pool.withdraw()` .
4. `@SubscriptionPool` sync user's virtual tracking token balance with latest `@AToken`
balance after supplying by calling `_syncUserBalance` .
```

RECOMMENDATION

Implement the `withdraw` function in the `SubscriptionPool` contract to allow users to withdraw their assets from the pool. Ensure that the function is well-documented and tested.

UPDATES

- *Apr 21, 2025*: The issue has been acknowledged by the Colend team.

2.2.3. DOS in `subscribe` logic in `SoulboundToken` contract LOW

Affected:

- `src/SouldboundToken.sol`#subscribe()

The `subscribe` function only allows the `onBehalfOf` address to represent each asset for a single user. This means that if a user wants to subscribe, an attacker can front-run the transaction and call `subscribe` on behalf of the user. This can result in a denial-of-service (DoS) attack, where the user is unable to subscribe because the `onBehalfOf` address has already been taken by the attacker.

```
 function subscribe(
     address asset,
     address onBehalfOf
 ) external notBlacklisted(onBehalfOf) doesNotOwnToken(onBehalfOf, asset) nonReentrant {
     require(allowedAsset[asset], "Must subscribe for a whitelisted asset");
     require(
         clndToken.transferFrom(msg.sender, address(this), tokenCost),
         "CLND transfer failed"
     );

     _mintToken(onBehalfOf, asset, msg.sender, "");
 }

modifier doesNotOwnToken(address onBehalfOf, address asset) {
```

```
    require(!userTokens[onBehalfOf][asset].exists, "Already owns a token"); //if attacker front-run the user transaction, the user will be unable to subscribe
    _;
}

function _mintToken(address to, address asset, address caller_, string memory tokenURI_) internal {
    _currentTokenId++;
    uint256 newTokenId = _currentTokenId;
    uint256 expiryTimestamp = block.timestamp + subscriptionDuration;

    userTokens[to][asset] = Token({
        tokenId: newTokenId,
        expiryTimestamp: expiryTimestamp,
        tokenAsset: asset,
        caller: caller_,
        paidMonths: 1,
        exists: true,
        tokenURI: tokenURI_
    });

    // Mark token as locked (soulbound)
    _locked[newTokenId] = true;
    // Store direct tokenId mappings
    _tokenURIs[newTokenId] = tokenURI_;
    _tokenOwners[newTokenId] = to;

    emit Locked(newTokenId);
    emit TokenMinted(to, newTokenId, expiryTimestamp, asset, tokenURI_);
}
```

RECOMMENDATION

To mitigate this issue, consider implementing a mechanism to allow users to subscribe with `onBehalfOf` address.

UPDATES

- *Apr 21, 2025*: The issue has been acknowledged by the Colend team.

2.2.4. `token.caller` can be overrided in `SouldboundToken` contract LOW

Affected:

- `src/SouldboundToken.sol`#extendMySubscription()

In the following functions, the `onBehalfOf` address can call the `extendMySubscription` function and pay the subscription fee to override the current `token.caller` address. If the original

`token.caller` had a long-term subscription for the `onBehalfOf` address, this override will result in the loss of the original `token.caller` information.

Currently, the `token.caller` is not critical to certain features. However, in future developments, if contracts rely on its information, this could lead to issues.

```
function extendMySubscription(
        address asset,
        address onBehalfOf
    )
        external
        notBlacklisted(onBehalfOf)
        tokenExists(onBehalfOf, asset)
        nonReentrant
    {
        Token storage token = userTokens[onBehalfOf][asset];

        if (token.caller == msg.sender) {
            require(
                clndToken.transferFrom(
                    msg.sender,
                    address(this),
                    tokenCost[asset]
                ),
                "CLND transfer failed"
            );
        } else if (onBehalfOf == msg.sender) {
            require(
                clndToken.transferFrom(
                    msg.sender,
                    address(this),
                    tokenCost[asset]
                ),
                "CLND transfer failed"
            );
            // update the caller in case the owner of the token is paying for himself now
            token.caller = msg.sender; //the token.caller is override by the onBehalfOf
address if the onBehalfOf address pay fee to call this function
        } else {
            revert("Unauthorized access");
        }
    }
```

2.2.5. Recommendation for handling unused parameters in `VirtualRewardTrackingToken` contract LOW

It is recommended to adopt a more idiomatic Solidity pattern by directly using the revert statement and marking unused parameters with inline comments:

```
function transfer(address /*to*/, uint256 /*value*/) public override(ERC20, IERC20) returns
(bool) {
    revert TransferNotAllowed();
}
```

This change improves code clarity and readability by removing unnecessary no-op expressions. It also clearly communicates that the parameters are intentionally unused and that the function is not meant to be executed.

UPDATES

- *Apr 21, 2025*: The issue has been acknowledged by the Colend team.

2.2.6. Unsafe transfer token INFORMATIVE

Affected:

- `src/SouldboundToken.sol`

The contract uses the `transfer` and `transferfrom` functions to transfer in/out tokens. Although there are required statements, `SafeERC20` is recommended to be used to handle more cases, as convenience is already ensured.

RECOMMENDATION

Use the `SafeERC20` library to handle token transfers.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 21, 2024*	Private Report	Verichains Lab

Table 2. Report versions history