verichains

SECURITY AUDIT OF

COLEND LOOP CORE SMART CONTRACT

Public Report

May 05, 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.
Solc	A compiler for Solidity.
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 May 05, 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 Loop Core Smart Contract. 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 a vulnerable issue in the smart contracts code. The vulnerability was fixed by the development team.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY 5

1.1. About Colend Loop Core Smart Contract 5

1.2. Audit Scope 5

1.3. Audit Methodology 5

1.4. Disclaimer 7

1.5. Acceptance Minute 7

2. AUDIT RESULT 8

2.1. Overview 8

2.2. Findings 8

2.2.1. [LOW] Missing Slippage Protection 8

3. VERSION HISTORY 10

1. MANAGEMENT SUMMARY

1.1. About Colend Loop Core Smart Contract

The Colend Loop Core Smart Contract is designed to implement a sophisticated looping strategy utilizing the forked Aave V3 lending protocol on the Core blockchain. This product enables users to leverage their CORE token positions by repeatedly borrowing and supplying tokens, thereby maximizing their returns through a process known as yield farming. By continuously cycling through borrowing and supplying, users can compound their interest and increase their overall yield. The integration with the Aave V3 protocol ensures that users benefit from the latest advancements in decentralized finance (DeFi) technology, providing a secure and efficient way to enhance their investment strategies.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of Colend Loop Core Smart Contract.

It was conducted on commit `fd507387c58c0b5f378ea3e648dcd271515c00a1` from the GitHub repository *https://github.com/tobyColend/colend-loop-core.git*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
`11c9ce45bc3dabeeb941435c33bdcc96aa78c38c9876cc08fa2bf677ac1ba2ae`	`src/AaveLoop.sol`
`4713a928ae06955cde41f5743cc77805e67e4310f8a6c004009b81fdeb35db0a`	`src/interfaces/IAaveLoop.sol`
`1148ac0d8ac30b448e24f87df8474ed6f7002b7ba2852ff4997c4d339b34e63f`	`src/interfaces/IEarn.sol`
`89ee99cbe11aa5b6a751e15684cefb010d2981e9eda32081b5c9a485b97023fe`	`src/interfaces/ISTCore.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.

2. AUDIT RESULT

2.1. Overview

The `Colend Loop Core Smart Contract` was developed using the `Solidity` programming language, specifically version `>=0.8.24`. The primary contract, `AaveLoop`, inherits from the `Ownable` and `ReentrancyGuard` libraries. This inheritance provides the contract with owner management capabilities and protection against reentrancy attacks. The `AaveLoop` contract allows users to leverage their CORE positions through a lending protocol similar to AaveV3, operating on the Core blockchain.

2.2. Findings

#	Title	Severity	Status
1	Missing Slippage Protection	LOW	FIXED

Table 2. Findings of the audit

2.2.1. [LOW] Missing Slippage Protection

Affected

- `src/AaveLoop.sol`#unloop()
- `src/AaveLoop.sol`#algebraSwapCallback()

Description

The swap operation in the `unloop()` function is vulnerable to sandwich attacks because it doesn't check for a minimum output amount.

```
function unloop() external nonReentrant { //missing slippage protection
    //...
    bool zeroForOne = i_stCore < i_wCore_Glyph;
    IUniswapV3Pool(i_swapPool_stCore_wCore).swap(
        address(this),
        zeroForOne,
        int256(aStCoreBalance),
        zeroForOne ? MIN_SQRT_RATIO + 1 : MAX_SQRT_RATIO - 1,
        abi.encode(data)
    );
    //...
}
```

RECOMMENDATION

- Add a `minAmountOut` parameter to the `unloop()` function

- Include this parameter in the `SwapCallbackData` struct
- Check in the `algebraSwapCallback` that the received amount meets the minimum threshold
- Revert the transaction if the slippage is too high

Example:

```
struct SwapCallbackData {
    address user;
    uint256 aStCoreBalance;
    uint256 minAmountOut;
}

function unloop(uint256 minAmountOut) external nonReentrant {
    //...

    SwapCallbackData memory data = SwapCallbackData({
        user: msg.sender,
        aStCoreBalance: aStCoreBalance,
        minAmountOut: minAmountOut
    });

    //...
}

function algebraSwapCallback(int256 amount0Delta, int256 amount1Delta, bytes calldata
_data) external {
    //...
    SwapCallbackData memory data = abi.decode(_data, (SwapCallbackData));

    uint256 wCoreBalance = IERC20(i_wCore_Glyph).balanceOf(address(this));

    // Check minimum amount received to prevent sandwich attacks
    require(wCoreBalance >= data.minAmountOut, "Slippage: insufficient output amount");

    //...
}
```

UPDATES

- ***March 19, 2025***: The development team has acknowledged and fixed the issue.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 19, 2025*	Public Report	Verichains Lab
1.1	*May 05, 2025*	Public Report	Verichains Lab

Table 3. Report versions history