SECURITY AUDIT OF

VIRIDIAN SMART CONTRACT

Public Report

May 17, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Coredao	Build with EVM-compatible smart contracts on a Bitcoin-powered blockchain
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.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on May 17, 2024. We would like to thank the Viridian 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 Viridian 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 did not identify vulnerable issue in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Viridian Smart Contract

The product combines the best of Uniswapv2 and Curve in a single AMM (Automated Market Maker). Supported by an Multi-chain interface that enables the project's scalability to new ecosystems at a high rate.

1.2. Audit scope

The audit focused on identifying security flaws in the *Viridian-Labs/contracts* repository on the branch *Launch*, which has a head commit `2beac283ef98b8a38fdb85c96692775066ec1215`, is forked from the repository *equilibre-finance/contracts* at the commit *549b7c4eeaaeaa448c6aa5d49c3ef30bbc7343f6*.

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

Viridian 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. Viridian understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Viridian 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 Viridian will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Viridian, the final report will be considered fully accepted by the Viridian without the signature.

2. AUDIT RESULT

2.1. Overview

The updated version focused on the following contracts:

2.1.1. Minter contract

The `update_period` function within the `Minter` contract is designed to be called once a week. It calculates the new active period, the weekly emission, the team's share of the emission, and the required balance.

The function then transfers the team's share to the team's address and updates the token balance and total supply in the rewards distributor.

A new version of code updating emission each week includes `EMISSION` to `97500/100000` and `TAIL_EMISSION` of a circulating emission to `1/100000`.

The `weekly` state, which is changed to `100_000 x1e18`, represents the starting weekly emission of the `VIRI` token.

A period lock on voting escrow tokens from 4 years to 1 year.

The active period changes from 2 weeks to 1 week.

The `calculate_growth` function, which calculates inflation, is updated to `0`.

2.1.2. VotingEscrow.sol

The `VoteingEscrow` contract is designed to lock tokens for a certain period of time. Voting escrow to have time-weighted votes. Votes have a weight depending on time, so users are committed to the future of (whatever they are voting for). The weight in this implementation is linear, and the lock cannot be more than `MAXTIME`. The updated version changes the maximum time to lock down to 1 year.

2.1.3. PairFactory

The `PairFactory` contract is used to create a new pair of two tokens. The fee of the factory is set at 0.3% for `volatileFee` and `0.04%` for `stableFee`. The `MAX_FEE` is set to `0.5%.`

2.1.4. ClaimAll.sol

The `ClaimAll` contract allows users to claim all fees from all bribe contracts using a single transaction. This contract is used to subsidize the gas fees for the users that are in the auto-claim list. Parameters of this contract such as `ve`, `voter`, `pairFactory`, and `rewards` are passed from the initialization function.

2.1.5. Viri.sol

The `Viri` contract allows a minter and a redemption receiver to mints tokens for any user. A main feature of this contract is transfer tokens between users. At initialization, the contract mints `6_000_000 x1e18` tokens for the first recipient.

2.2. Findings

During the audit process, the audit team did not identify vulnerable issue in the contract code.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*May 17, 2024*	Public Report	Verichains Lab

Table 2. Report versions history