verichains

SECURITY AUDIT OF

VERA SMART CONTRACT

Public Report

May 13, 2022

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 prepared by Verichains Lab on May 13, 2022. We would like to thank the Vera Labs Inc 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 Vera 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 no vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Vera Smart Contract

Vera is a decentralized protocol built on top of major blockchains that allows essential financial services for NFTs such as renting, lending, and mortgages. Decentralized protocols are non-custodial, meaning digital assets and NFTs are not owned by any middleman or intermediary.

VERA is the native utility token for the platform used for governance and fee collections purposes.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Vera Smart Contract. It was conducted on the source code provided by the Vera Labs Inc team.

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 was 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

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The initial review was conducted on Jan 19, 2022 and a total effort of 7 working days was dedicated to identifying and documenting security issues in the code base of the Vera Smart Contract.

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

SHA256 Sum	File
2d466b469fd6079fdca2305b8716320d460c40e2060a6dc081f76bd0b223eab7	**./base/BasePool.sol**
4df1f949bfcddf7305dfba1ef6021842105ebd1c793a5c440217af60f69743b2	**./base/TokenSaver.sol**
bbc65efeb36fe3b5674fb7774270323aa9c1ad9a6d1517a206bdcfed648b3eea	**./base/AbstractRewards.sol**
65d66fa08c13c10901a59a4c3c19d9c1075396715491e8b4f48d659bf5dd77c5	**./TimeLockNonTransferablePool.sol**
7490e734dccc420440f73fda9faa95500e8a56c64ed38535788cd9a78bde4440	**./interfaces/IBasePool.sol**
e4d54710f7d465f7b264f10c571373c078dce11e2c677bf334220fcd97f68d0b	**./interfaces/IAbstractRewards.sol**
6ed743f409d47a1fd57d6cfd82c63a9d4780e18e92718eef0822c19eb47492ae	**./interfaces/ITimeLockPool.sol**
780b387f9c2639481484774bd6addf11a89359936fe17e06b237cc18077f02e7	**./View.sol**
cc36e4786ebc928814354f1cc6d43bac72b48c4d9102678daaf55f9f8dc263c7	**./TimeLockPool.sol**
32e20dc9c7834eb1abcb11483bc16b9467e6329d27fa4ea0f310bfdc854052c2	**./LiquidityMiningManager.sol**

2.2. Contract codes

The Vera Smart Contract was written in Solidity language, with the required version to be 0.8.7.

2.3. Findings

During the audit process, the audit team found no vulnerability in the given version of the Vera Smart Contract.

2.4. Additional notes and recommendations

2.4.1. LiquidityMiningManager.sol - No approval reset in removePool function INFORMATIVE

Manager approve _poolContract for using all the tokens in addPool:

```
function addPool(address _poolContract, uint256 _weight) external onlyGov…
    {
     distributeRewards();
     require(_poolContract != address(0), "LiquidityMiningManager.addPool:…
    pool contract must be set");
     require(!poolAdded[_poolContract], "LiquidityMiningManager.addPool: P…
   ool already added");
     require(pools.length < MAX_POOL_COUNT, "LiquidityMiningManager.addPoo…
   l: Max amount of pools reached");
     // add pool
     pools.push(Pool({
         poolContract: IBasePool(_poolContract),
         weight: _weight
     }));
     poolAdded[_poolContract] = true;

     // increase totalWeight
     totalWeight += _weight;

     // Approve max token amount
     reward.safeApprove(_poolContract, type(uint256).max);

     emit PoolAdded(_poolContract, _weight);
}
```

But there is no approval reset in removePool.

RECOMMENDATION

Reset the approval in removePool method.

```
// Approve max token amount
reward.safeApprove(_poolContract, 0);
```

UPDATES

- *May 13, 2022*: This issue has been acknowledged by the Vera Labs Inc team.

2.4.2. LiquidityMiningManager.sol - Conflicts access modifier onlyGov and onlyRewardDistributor INFORMATIVE

In LiquidityMiningManager, all methods with onlyGov modifiers call distributeRewards which is protected by onlyRewardDistributor, which means that gov must also be reward distributor so that these calls can be executed successfully.

The above guarantee is not stated within the contract's code, comments, or documents given to the audit team.

RECOMMENDATION

Use onlyGovOrRewardDistributor for distributeRewards.

UPDATES

- *May 13, 2022*: This issue has been acknowledged by the Vera Labs Inc team.

2.4.3. BasePool.sol - _rewardToken and _escrowPool should be checked before assignment INFORMATIVE

```
constructor(
    string memory _name,
    string memory _symbol,
    address _depositToken,
    address _rewardToken,
    address _escrowPool,
    uint256 _escrowPortion,
    uint256 _escrowDuration
) ERC20Permit(_name) ERC20(_name, _symbol) AbstractRewards(balanceOf, tot…
  alSupply) {
    require(_escrowPortion <= 1e18, "BasePool.constructor: Cannot escrow …
  more than 100%");
    require(_depositToken != address(0), "BasePool.constructor: Deposit t…
  oken must be set");
    depositToken = IERC20(_depositToken);
    rewardToken = IERC20(_rewardToken);
    escrowPool = ITimeLockPool(_escrowPool);
    escrowPortion = _escrowPortion;
    escrowDuration = _escrowDuration;

    if(_rewardToken != address(0) && _escrowPool != address(0)) {
        IERC20(_rewardToken).safeApprove(_escrowPool, type(uint256).max);
```

```
    }
}
```

RECOMMENDATION

The code can be fixed as below.

```
constructor(
    string memory _name,
    string memory _symbol,
    address _depositToken,
    address _rewardToken,
    address _escrowPool,
    uint256 _escrowPortion,
    uint256 _escrowDuration
) ERC20Permit(_name) ERC20(_name, _symbol) AbstractRewards(balanceOf, tot…
  alSupply) {
    require(_escrowPortion <= 1e18, "BasePool.constructor: Cannot escrow …
  more than 100%");
    require(_depositToken != address(0), "BasePool.constructor: Deposit t…
  oken must be set");
    depositToken = IERC20(_depositToken);
    require(_rewardToken != address(0) && _escrowPool != address(0), "INV…
  ALID PARAMETER");
    rewardToken = IERC20(_rewardToken);
    escrowPool = ITimeLockPool(_escrowPool);
    escrowPortion = _escrowPortion;
    escrowDuration = _escrowDuration;
    IERC20(_rewardToken).safeApprove(_escrowPool, type(uint256).max);
}
```

UPDATES

- *May 13, 2022*: This issue has been acknowledged by the Vera Labs Inc team.

2.4.4. TokenSaver.sol - Using deprecated function _setupRole INFORMATIVE

The _setupRole function is deprecated in favor of the _grantRole function.

```
constructor() {
    _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
}
```

RECOMMENDATION

Using the _grantRole function instead of _setupRole.

UPDATES

- *May 13, 2022*: This issue has been acknowledged by the Vera Labs Inc team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 26, 2022*	Private Report	Verichains Lab
1.1	*May 13, 2022*	Public Report	Verichains Lab

Table 2. Report versions history