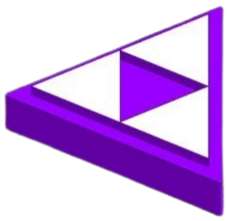
verichains

SECURITY AUDIT OF

DTRINITY



Public Report

Jun 17, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
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.
Ethereum	An opensource platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
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 Jun 17, 2024. We would like to thank the dTrinity team 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 dTrinity. 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 identify a vulnerable issue in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About

dTrinity is a DeFi super-protocol designed to provide sustainable yield, lower borrowing cost, and USD stablecoin liquidity for emerging L2 and L3 ecosystems, starting with Fraxtal.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of dTrinity. It was conducted on commit `e83dee81a65d99b5558eeb6b6a74c02611a883a5` from git repository link: *https://github.com/dtrinity/trinity-solidity-contracts.git*.

The protocol is forked from Uniswap V3 and Aave V3. The audit team reviewed the codebase of the origin code to ensure that the protocol is implemented correctly and securely:

Name	Repository	Commit Hash
Uniswap V3 Core	Uniswap/v3-core	`d8b1c635c275d2a9450bd6a78f3fa2484fef73eb`
Uniswap V3 periphery	Uniswap/v3-periphery	`697c2474757ea89fec12a4e6db16a574fe259610`
Uniswap solidity lib	Uniswap/solidity-lib	`c01640b0f0f1d8a85cba8de378cc48469fcfd9a6`
Aave V3 core	aave/aave-v3-core	`724a9ef43adf139437ba87dcbab63462394d4601`
Aave V3 periphery	aave-v3-periphery	`803c3e7d6d1c6da8d91411f4d085494f7189ea0b`

Table 1. List of forked commits

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 2. Severity levels

1.4. Disclaimer

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

2. AUDIT RESULT

2.1. Overview

dTrinity was developed using the `Solidity` language, with the required versions being `^0.7.0` and `^0.8.0`.

The audit focused on forking and updating from the `UniswapV3-core`, `UniswapV3-periphery`, `AaveV3-core`, `AaveV3-periphery`, and `Solidity-lib` repositories. The audit was conducted on a dTrinity's repository.

2.1.1. Dex Contracts

In the Dex contracts, the updated version introduces tick spacing for the fee amount. The fee protocol requirements have been modified to range from 0 to 10, compared to the previous range of 4 to 10. Protocol fees are collected from swap fees: a small portion of swap fees is subtracted and saved as protocol fees to later be collected by the Factory contract owner. It must be zero or between 1 and 1/10 of swap fees.

The initialization code has been updated to follow the changes made to the Uniswap V3 pool.

2.1.2. Lending Contracts

For the Oracle contract, the `base_currency_init` must match the `base_currency_init` of the fallback oracle when setting it.

The `priceFeedDecimals` and `rewardPriceFeed` are reset after obtaining reward information from the oracle.

A `depositReward()` function has been added to the `AaveOracle` interface. The `depositRewardFrom()` function has been added to the `rewardsController` contract, allowing users to send reward tokens to the reward controller contract to change the emission per second.

New `dswap` contracts have been added. The `dswap` adapter supports users in buying, selling, and flash-loaning tokens with the Uniswap V3 pool.

2.2. Findings

During the audit process, the audit team identify a vulnerable issue in the contract code.

2.2.1. HIGH - Arbitrary `transferfrom` calls in `depositRewardFrom` function

Position:

- `contracts/lending/periphery/rewards/RewardsController.sol#depositRewardFrom()`

Description: The protocol allows users to deposit more rewards into the rewards controller contract through the `depositReward` function in `EmissionManager` contract. The `depositRewardFrom` function in the `RewardsController` contract allows any address to call. This can be exploited by an attacker to force addresses that have approved the `RewardsController` contract to make an unintended contribution.

```
// contracts/lending/periphery/rewards/EmissionManager.sol
function depositReward(
    address asset,
    address reward,
    uint256 amount
) external {
    require(amount > 0, "ZERO_AMOUNT");
    _rewardsController.depositRewardFrom(asset, reward, amount, msg.sender);
}

// contracts/lending/periphery/rewards/RewardsController.sol
function depositRewardFrom(
    address asset,
    address reward,
    uint256 amount,
    address from
) external {
   //...

    try
        IPullRewardsTransferStrategy(transferStrategyAddress)
            .getRewardsVault()
    returns (address vault) {
        //...
        IERC20(reward).transferFrom(from, vault, amount);
        //...
    } catch {
        revert("ONLY_ALLOW_DEPOSIT_TO_PULL_REWARDS_TRANSFER_STRATEGY");
    }
}
```

2.2.1.1. Recommendation:

Add a `require` statement to ensure that the caller is the `EmissionManager` contract.

UPDATES:

- ***Jun 17, 2024***: The issue has been acknowledged and fixed by the dTrinity team.

3. VERSION HISTORY

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

Table 3. Report versions history