HashEx
BLOCKCHAIN SECURITY

TraderJoe Lending

smart contracts
final audit report

September 2021

hashex.org

contact@hashex.org

Contents

1. Disclaimer

2. Overview

HashEx was commissioned by TraderJoe team to perform an audit of their smart contracts.

The code located in the GitHub repository @traderjoe-xyz/joe-lending was audited after the commit 99e44ae. The repository contains tests for major contracts, but the last time they were updated was in the 3d03d86 commit. The code was provided without documentation besides the Compound Finance docs [1] and C.R.E.A.M. Finance docs [2].

The audited project is the fork of C.R.E.A.M. Finance lending contracts (audited by Trail of Bits in 2021, public report isn't available) which were forked from Compound Finance (audited by OpenZeppelin and Trail of Bits multiple times since 2019, list of audits is available [3]), with additional parts forked from BENQI Finance (audited by Halborn in 2021 [4]).

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts.
- Formally check the logic behind given smart contracts.

Information in this report should be used to understand the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

2.1 Summary

Project name	TraderJoe Lending
URL	https://www.traderjoexyz.com
Platform	Avalanche Network
Language	Solidity

2.2 Contracts

Name	Address
Issues for multiple subjected contracts	https://github.com/traderjoe-xyz/joe-lending/tree/99e44aec2d69bbd15c0afc2fceb873723f979706/contracts
Comptroller	https://github.com/traderjoe-xyz/joe-lending/blob/99e44aec2d69bbd15c0afc2fceb873723f979706/contracts/Comptroller.sol
CCapableErc20	https://github.com/traderjoe-xyz/joe-lending/blob/99e44aec2d69bbd15c0afc2fceb873723f979706/contracts/CCapableErc20.sol
CTokenDeprecated	https://github.com/traderjoe-xyz/joe-lending/blob/99e44aec2d69bbd15c0afc2fceb873723f979706/contracts/CTokenDeprecated.sol
COMP	https://github.com/traderjoe-xyz/joe-lending/blob/99e44aec2d69bbd15c0afc2fceb873723f979706/contracts/Governance/Comp.sol
CompoundLens	https://github.com/traderjoe-xyz/joe-lending/blob/99e44aec2d69bbd15c0afc2fceb873723f979706/contracts/Lens/CompoundLens.sol
TripleSlopeRateModel	https://github.com/traderjoe-xyz/joe-lending/blob/99e44aec2d69bbd15c0afc2fceb873723f979706/contracts/TripleSlopeRateModel.sol

3. Found issues

12
Total issues

●	High	1 (8%)
●	Medium	3 (25%)
●	Low	7 (58%)
●	Info	1 (9%)

C1. Issues for multiple subjected contracts

ID	Severity	Title	Status
C1-01	● High	Owner/admin can break contracts or steal funds in numerous ways	Open
C1-02	● Low	Use of assert() function	Open
C1-03	● Low	Native transfers	Open
C1-04	● Info	General recommendations	Acknowledged

C2. Comptroller

ID	Severity	Title	Status
C2-01	● Medium	Credit limit accounts	Open
C2-02	● Low	Missing events	Open

C3. CCapableErc20

ID	Severity	Title	Status
C3-01	● Medium	Flash loan permission	? Open

C4. CTokenDeprecated

ID	Severity	Title	Status
C4-01	● Medium	Doesn't support AVAX network	? Open

C5. COMP

ID	Severity	Title	Status
C5-01	● Low	Wrong parameters	? Open
C5-02	● Low	Unconventional naming	? Open

C6. CompoundLens

ID	Severity	Title	Status
C6-01	● Low	Wrong parameter	? Open
C6-02	● Low	No view keywords	✓ Acknowledged

4. Contracts

C1. Issues for multiple subjected contracts

Issues

C1-01 Owner/admin can break contracts or steal funds in numerous ways ● High ⓘ Open

TraderJoe protocol uses the deploy scheme that was first implemented by Compound Finance, but unlike Compound, there are no governance contracts in the audited repository. Thus one can conclude that the TraderJoe team plans to introduce an ownership scheme from C.R.E.A.M. Finance, i.e. centralized administration behind the Timelock contract.

CToken

`_setReserveFactor()` function L877 updates the `reserveFactorMantissa` variable up to 100%, regulated by `reserveFactorMaxMantissa` in CTokenInterface. Setting it to 100% would distribute all the dividends into the reserves.

`_reduceReserves()` function L982 transfers reserves to admin.

`_setInterestRateModel()` function L1046 could be used to set malicious rate model contract with broken `getBorrowRate()` parameters causing the permanent failure of `accrueInterest()` function of CToken contract.

JumpRateModelV2 & TripleSlopeRateModel

`updateJumpRateModel()` L94 and `updateTripleRateModel()` L102 allows the owner of the rate model contract to change the parameters without any restrictions. Setting unrealistically wrong parameters could break the interaction using this model CToken contracts.

Comptroller

`_setLiquidationIncentive()`, `_setCloseFactor()` and `_setCollateralFactor()` functions in L1115, L1054 and L1074 have no checks on input values, leading to the possibility of broken liquidation process or stealing users' funds.

`_setPauseGuardian()` function in L1274 updates the `pauseGuardian` address, who can pause any single process of the system.

`updateCTokenVersion()` function in L756 could break interaction with the CToken contract if the wrong version is set.

`_setCompSpeed()` function in L1646 has no checks on input `compSpeed` value causing the possible minting of the arbitrary amount of rewards in a single block and attack on the liquidity pools by the malicious admin of the Comptroller.

`_setCreditLimit()` and `_supportMarket()` functions in L1368 and L1140 allow the owner to set an arbitrary address as a listed market or give them unlimited credit.

All these examples can't be eliminated even if the ownership is transferred to the Timelock contract (with a minimum delay of 48 hours). The original Compound project implements the governance model (Governor Alpha and Bravo) that could be monitored with Compound Lens. At the current state, users have no choice but to trust the owners of the audited contracts.

C1-02 Use of assert() function ● Low ⓘ Open

In the contract Comptroller (functions `exitMarket()` L181 and `borrowAllowed()` L399), Exponential (function `mulExp()` L197) and V1PriceOracle (functions `calculateSwing()` L829 and `capToMax()` L846) the function `assert()`is used, but it is not recommended. It is better to use the `require()` function.

C1-03 Native transfers ● Low ⓘ Open

`grantRewardInternal()` function in L1607 of Comptroller and `doTransferOut()` in L136 of CEther contract use the `transfer()` method of sending native currency that is now discouraged due to non-flexible gas management. The recommended function is `call()` with an additional reentrancy guard.

C1-04 General recommendations ● Info ✓ Acknowledged

In the project, the 0.5.16 version of the compiler is used. We also recommend fixing the pragma version.

C2. Comptroller

Overview

Main control contract that keeps the registry of valid CTokens and supports their interaction. Inherited from ComptrollerV1Storage, ComptrollerInterface, ComptrollerErrorReporter and Exponential contracts.

Issues

C2-01 Credit limit accounts ● Medium ⓘ Open

`_setCreditLimit()` function in L1368 sets the credit limit for the account that replaces the collateral in `getHypotheticalAccountLiquidity()` checks. On the other hand, the credit account's collateral would be ignored during these checks.

C2-02 Missing events ● Low ⓘ Open

In the `setJoeAddress()` function L1655 there is no appropriate event.

C3. CCapableErc20

Overview

The description from the comments in code: deprecated Cream's CCapableErc20 Contract.

Issues

C3-01 Flash loan permission ● Medium ⓘ Open

Unlike CCollateralCapErc20, `flashLoan()` function in L136 of CCapableErc20 contract doesn't call the Comptroller for approval. We believe that the CCapableErc20 contract should be marked as deprecated and not intended to be deployed.

C4. CTokenDeprecated

Overview

The description from the comments in code: deprecated CToken Contract only for CEther.

Issues

C4-01 Doesn't support AVAX network ● Medium ⓘ Open

In the function `getBlockNumber()` L219, `block.number` is used, but in all the other contracts in the same function `block.timestamp`is used. We believe that the CCapableErc20 contract is not intended to be deployed.

C5. COMP

Overview

Reward ERC20 token with governance.

Issues

C5-01 Wrong parameters ● Low ⓘ Open

COMP governance token has its name and symbol variables from the source of the fork, see L8-11.

C5-02 Unconventional naming ● Low ⓘ Open

`name`, `symbol`, `decimals` and `totalSupply` variables are declared constants but named in mixedCase instead of UPPERCASE style.

C6. CompoundLens

Overview

Contract aggregating view functions and parameters.

Issues

C6-01 Wrong parameter ● Low ⓘ Open

CompoundLens contract specifically checks the input `cToken.symbol` to be equal to `crFTM` in L61, L145. We believe it wasn't changed after the fork of C.R.E.A.M. Finance.

C6-02 No view keywords ● Low ✓ Acknowledged

There're functions of the CompoundLens contract that could be marked as ‘view'.

C7. TripleSlopeRateModel

Overview

Interest rate model by C.R.E.A.M. Finance.

5. Conclusion

1 high severity issue was found. The contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on the code improving and preventing potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

8. References

1. Compound Finance docs
2. C.R.E.A.M. Finance docs
3. Compound audits list
4. BENQI audit by Halborn

contact@hashex.org

@hashexbot

blog.hashex.org

linkedin

github

twitter

#HashEx
BLOCKCHAIN SECURITY