HashEx
BLOCKCHAIN SECURITY

Deepr Finance

smart contracts
final audit report

November 2022

hashex.org
contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Deepr Finance team to perform an audit of their smart contract. The audit was conducted between 12/10/2022 and 18/10/2022.

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 for understanding 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.

The code is available in the @Deepr-Finance/deepr-protocol GitHub repository after the commit a87400c.

The audited contracts are mainly forked from Compound Finance (@compound-finance/compound-protocol and @compound-finance/open-oracle Github repositories), which has public audits available.

Audited contracts are designed to be deployed behind proxies. Users need to check the current system admin before interacting with the contacts.

Update. The Deepr Finance team has responded to this report. The updated code is located in the same repository after the commit 55f0535.

2.1 Summary

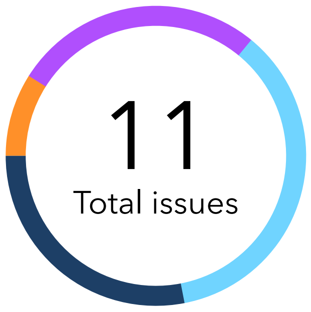
Project name	Deepr Finance
URL	https://deepr.finance

Platform	Shimmer Network
Language	Solidity

2.2 Contracts

Name	Address
Comp	
Governors	
CompoundLens	
OpenOracle	
Comptroller	
LinearRateModel	
BaseJumpRateModelV2 & WhitePaperInterestRateModel	
Timelock	
Compound protocol contracts	

3. Found issues



●	High	1 (9%)
●	Medium	3 (27%)
●	Low	4 (36%)
●	Info	3 (28%)

C2. Governors

ID	Severity	Title	Status
C2-01	● Medium	Ethereum based constants	✓ Resolved

C4. OpenOracle

ID	Severity	Title	Status
C4-01	● Low	Gas optimizations	✓ Resolved

C5. Comptroller

ID	Severity	Title	Status
C5-01	● High	Exaggerated owner's rights	✓ Partially fixed
C5-02	● Medium	Rewards distribution differs from documentation	✓ Resolved

C5-03	● Low	Gas optimizations	✓ Resolved
C5-04	● Low	Unreliable cToken verification	✓ Resolved
C5-05	● Info	Typos	✓ Resolved

C6. LinearRateModel

ID	Severity	Title	Status
C6-01	● Low	Gas optimizations	✓ Resolved
C6-02	● Info	Undefined behavior outside the specified period	✓ Resolved
C6-03	● Info	Typos	✓ Resolved

C7. BaseJumpRateModelV2 & WhitePaperInterestRateModel

ID	Severity	Title	Status
C7-01	● Medium	Ethereum based constants	✓ Resolved

4. Contracts

C1. Comp

Overview

Governance token, a fork of the COMP token with a different total supply of 1 billion with 18 decimals. Voting system relies on timestamps instead of block numbers.

No issues were found.

C2. Governors

Overview

Governance system forked from Compound Finance protocol.

Contracts: GovernorAlpha, GovernorBravoDelegate, GovernorBravoDelegator, GovernorBravoEvents.

Issues

C2-01 Ethereum based constants ● Medium ✓ Resolved

Governors' thresholds are directly forked from Compound, but Deepr token has a total supply that's 100 times bigger.

Governance must be established with caution with taking the actual voting power into consideration.

Affected contracts: GovernorAlpha, GovernorBravoDelegate.

C3. CompoundLens

Overview

Compound Lens fork. No issues were found.

C4. OpenOracle

Overview

Open Oracle contracts forked from Compound Finance repository. Public audit of the early-stage version was audited by OpenZeppelin.

Contracts: OpenOracleData, OpenOraclePriceData, OpenOracleView, UniswapAnchoredView, UniswapConfig, UniswapV2OracleLibrary.

Issues

C4-01 Gas optimizations ● Low ✓ Resolved

The checked math `mul()` function should be reduced to `return a * b;` or the operations have to be moved inside the `unchecked` section. Solidity enables over- and underflow checks by default after the 0.8 version.

C5. Comptroller

Overview

The main contract of Deepr protocol forked from Compound Finance. The risk model contract, which validates permissible user actions and disallows actions if they do not fit certain risk parameters. For instance, the Comptroller enforces that each borrowing user must maintain a sufficient collateral balance across all cTokens.

The new feature introduced by Deepr Finance is the rewards distribution system across markets and suppliers/borrowers. The ratio for splitting COMP rewards between users and project owners is hardcoded as 77 to 23 (it could be changed with the proxy's implementation upgrade).

Issues

C5-01 Exaggerated owner's rights ● High Partially fixed

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

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

The `_setLinearRateModelAddress()` function allows the admin to control COMP rewards distribution, possibly minting the arbitrary number of rewards in a single block and attack on the liquidity pools.

The `_supportMarket()` function allows the owner to set an arbitrary address as a listed market and drain the valuable assets.

The `_setNewNextEpochTimestamp()` function could be used for blocking the automatic updates of reward rate in the `refreshCompSpeeds()` by setting the `nextEpochTimestamp` variable in distant future.

There are also some examples of overpowered ownership in CToken and BaseJumpRateModelV2 contracts:

`updateJumpRateModel()` 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.

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

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

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

Recommendation

We recommend securing the governance system as much as possible. Admin must be a Timelock contract with at least 48 hours of minimum delay, its administration should be accredited to a MultiSig or community governance.

C5-02 Rewards distribution differs from documentation ● Medium ✓ Resolved

COMP rewards distribution slightly differs from the whitepaper: with each update of the epoch, both users and the team will receive slightly more than planned. In this case, the rewards will be used up earlier than the expected time.

```
    function refreshCompSpeeds() public {
        ...
        refreshCompRateInternal();
        refreshCompSpeedsInternal();

        updateTeamAllocation();
        refreshTeamCompSpeed();
        ...
    }

    function refreshCompRateInternal() internal {
        compRate = linearRateModel.getActualRate();
        ...
    }

    function refreshCompSpeedsInternal() internal {
        ...
        for (uint i = 0; i < allMarkets_.length; i++) {
            ...
            updateCompSupplyIndex(address(cToken));
            updateCompBorrowIndex(address(cToken), borrowIndex);
        }
        ...
        uint userCompRate = mul_(compRate, Exp({ mantissa: 0.77e18 }));
        for (uint i = 0; i < allMarkets_.length; i++) {
            ...
            compSpeedsPerBlock[address(cToken)].supply = newSupplySpeed;
            compSpeedsPerBlock[address(cToken)].borrow = newBorrowSpeed;
        }
    }

    function updateTeamAllocation() public {
        uint compSpeed = compTeamSpeed;
        uint deltaTimestamp = sub_(timestamp, lastTeamTimestamp);
        ...
        uint newAccrued = mul_(deltaTimestamp, compSpeed);
        uint teamAccrued = add_(compAccrued[teamAddress], newAccrued);
        compAccrued[teamAddress] = teamAccrued;
    }

    function refreshTeamCompSpeed() internal {
        compTeamSpeed = mul_(compRate, Exp({ mantissa: 0.23e18 }));
        ...
    }
```

Also, rate update via `refreshCompSpeeds()` should be performed first hand in the `_setWeightMarkets()` and `_setWeightBorrowMarkets()` functions, otherwise parameters would be updated retrospectively.

```
    function _setWeightMarkets(...) external {
        ...
        refreshCompSpeeds();
    }
```

```
function _setWeightBorrowMarkets(...) external {
    ...
    refreshCompSpeeds();
}
```

Recommendation

Conform with the documentation by handling the custom timestamp (including the possibility of skipped epochs).

Update the `_setWeightMarkets()` and `_setWeightBorrowMarkets()` functions and make a call of the `refreshCompSpeeds()` function in the beginning:

```
function _setWeightMarkets(...) external {
    refreshCompSpeeds();
    ...
}

function _setWeightBorrowMarkets(...) external {
    refreshCompSpeeds();
    ...
}
```

C5-03 Gas optimizations ● Low ✓ Resolved

1. Reading of `allMarkets.length` in `for()` loop should be avoided.
2. `refreshCompSpeedsInternal()` could avoid gas spending on updating only `isListed` markets.
3. Once an epoch unnecessary calls of `updateCompSupplyIndex()` or `updateCompBorrowIndex()` immediately after calling the `refreshCompSpeeds()`. Affected functions: `mintAllowed()`, `redeemAllowed()`, `borrowAllowed()`, `repayBorrowAllowed()`, `seizeAllowed()`, and `transferAllowed()`. The `refreshCompSpeeds()` function may return a boolean result to avoid duplicated CompIndex updates.

C5-04 Unreliable cToken verification   ● Low   ✓ Resolved

The `_setWeightMarkets()` function could check actual status of updating markets instead of try/catch `cToken` methods.

```
    function _setWeightMarkets(address[] calldata cTokens, uint[] calldata weights)
 external {
        ...
        for (uint i = 0; i < cTokens.length; i++) {
            try CToken(cTokens[i]).isCToken() returns (bool) {
                ...
            } catch {
                revert("cToken is not a valid CToken");
            }
        }
        ...
    }
```

For example, it could require `markets[cTokens[i]].isListed`.

Also, in the current realization a possible return of a `false` value in `isCToken()` is not handled properly, as well as reverting should be done with specific `cToken` address, which caused the failure.

C5-05 Typos   ● Info   ✓ Resolved

Typos reduce the code's readability: 'weigths', 'addres'.

C6. LinearRateModel

Overview

Negative slope linear model for speed of rewards distribution for Comptroller contract.

Described in a separate documentation file provided by the team: it's a linear function in RatePerEpoch-vs-Epoch coordinates, making it a piece-wise function in Rate-vs-Timestamp coordinates. Slope is defined by initial rate value, epoch duration, and the total number of epochs.

Issues

C6-01 Gas optimizations ● Low ✓ Resolved

1. `getRatePerEpoch()` should recalculate the `slope` value from `xIntersect` and `yIntersect` instead of reading it from the storage.
2. Multiple reads from storage of the `initTimestamp` variable in `getActualEpoch()`, `getNextEpochTimestamp()`, and `getEpochAndRate()` could be avoided by using local variables.
3. In the function `_setNewParameters()` the global variables `yIntersect` and `xIntersect` are read after writing.
4. In the function `_setNewAdmin()` global variable `admin` is read after writing.

C6-02 Undefined behavior outside the specified period ● Info ✓ Resolved

The rate model is implicitly defined as zeroes out of a specified time period from starting point `initTimestamp` to the end of the last epoch `initTimestamp + epochTimestamp * totalOfEpochs_`. However, the `getActualEpoch()` function explicitly denies reading the rate for timestamps prior the `initTimestamp`:

```
    function getActualEpoch() public view returns (uint) {
        require(
            initTimestamp < block.timestamp,
            "The timestamp is higher than the initTimestamp, wait till the epoch 0
starts."
```

```
        );
        return (block.timestamp - initTimestamp) / epochTimestamp;
    }
```

C6-03 Typos ● Info ✓ Resolved

Typos reduce the code's readability: 'purpouses', 'tiemstamp'.

C7. BaseJumpRateModelV2 & WhitePaperInterestRateModel

Overview

Rate model contracts forked from Compound Finance repository.

Issues

C7-01 Ethereum based constants ● Medium ✓ Resolved

Global variable `blocksPerYear` is directly forked from Compound, but Shimmer Network doesn't have fixed blocktime.

C8. Timelock

Overview

Original Timelock contract forked from Compound Finance with a minimum delay of 2 days. Audit by OpenZeppelin is available.

No issues were found.

C9. Compound protocol contracts

Overview

Contracts of the lending system (apart from the Comptroller contract) forked from Compound Finance. Security audits are available.

No issues were found.

5. Conclusion

1 high, 3 medium, 4 low severity issues were found during the audit. 3 medium, 4 low issues were resolved in the update.

The reviewed contract is extremely dependent on the owner’s account. The contract is designed to be deployed with proxies and contains multiple functions able to halt transfers. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on improving the code 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

contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx
BLOCKCHAIN SECURITY