HashEx

Twin Finance

smart contracts audit report

Prepared for:

twin.finance

Authors: HashEx audit team

July 2021

Contents

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.

DISCLAIMER: 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, 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.

Introduction

HashEx was commissioned by the Twin Finance team to perform an audit of their smart contracts. The audit was conducted between June 24 and July 01, 2021.

The code located in the @twin-finance/contracts Github repository was audited after the `69f3aae` commit. Documentation can be found on the twin website.
The TWIN token is deployed to Binance Smart Chain (BSC):
TWIN.sol `0x62907AD5c2D79e2A4F048a90Ae2B49d062a773f3`.

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.

Update: Twin Finance team has responded to this report. Individual responses were added after each item in the section. The updated code is located in the same repository after the `6f61b0a` commit.

Updated contracts are deployed to BSC at:

TWIN	`0x62907AD5c2D79e2A4F048a90Ae2B49d062a773f3`,
Minter	`0x4cA7eA2a21230dF34B8b7C3576B0790aEa2F8877`,
MasterChefV2	`0x6287fFd0f2fFF3ac7Bc39751ed138188B83bB711`,
TFundOracle	`0x072C52006784163C406a0031051Bf8b5eE184383`,
TAsset	`0x79E29E397f33A4F515c11b913e5357327AE14aEC`,
	`0xc40bdAA708dA1446c13a82a5AFc18A396f40Bb40`,
	`0x7191d810103d34EEc28E04c61298551f2381fa6e`,
ChainLinkLikeOracle	`0xc6B22438f927b5C3bb70d138dABF6dcC590CaAdB`,
CDPToken	`0x1537a004D9f46a835Baf4a05c0dE1CbbB93e1B32`,
OracleManager	`0xa62DfAb32563c742FC61dd8B4c68095346DD3AA0`.

Contracts overview

`MasterChefV2.sol`

Farming contract, variation of MasterChef by SushiSwap [1,2].

`TWIN.sol`

Mintable ERC20 token.

`TAsset.sol`

Mintable ERC20 token.

`CDPToken.sol`

ERC721 token with custom metadata storage.

`ChainLinkLikeOracle.sol`

Simple oracle contract with a price to be set directly by the owner.

`OracleManager.sol`

Oracle contract gathering data from multiple ChainLinkLikeOracle oracles.

`TFundOracle.sol`

Oracle aggregator gathering data from multiple ChainLinkLikeOracle oracles.

`Minter.sol`

The main contract controlling the minting process of TAssets and CDPTokens.

Found issues

ID	Title	Severity	Response
01	TWIN: mint is open for accounts with farm role	High	Responded
02	TWIN: vesting not working	High	Responded
03	Minter: centralized price feed	High	Responded
04	MasterChefV2: no cap for twinPerBlock	High	Fixed
05	Minter: division before multiplication	Medium	Fixed
06	Minter: relying on ERC20 implementation	Medium	Fixed
07	MasterChefV2: minting capped token	Medium	Informed
08	TFundOracle: for() loop gas problem	Medium	Fixed
09	TWIN: input data not checked	Low	Informed
10	TWIN: variables naming	Low	Informed
11	Minter: unindexed events	Low	Fixed
12	Minter: duplicated events	Low	Fixed
13	Minter: typos in error messages	Low	Fixed
14	Minter: input data not checked	Low	Fixed
15	Minter: CollateralToken gas optimization	Low	Fixed
16	Minter: variables naming	Low	Informed
17	Minter: TODO comment	Low	Fixed
18	CDPToken: unindexed events	Low	Fixed
19	ChainLinkLikeOracle: only for decimals <= 18	Low	Fixed
20	OracleManager: empty constructor	Low	Fixed

21	OracleManager: typos in function name	Low	Fixed
22	MaterChefV2: input data not checked	Low	Informed
23	MaterChefV2: variables naming	Low	Informed
24	MaterChefV2: redundant code	Low	Fixed
25	MaterChefV2: typos in comments	Low	Fixed
26	General recommendations	Low	Fixed

#01 TWIN: mint is open for accounts with farm role High

TWIN is a mintable ERC20 standard [3] token with `mint()` and `burn()` function accessible for the addresses with the FARM_ROLE. While the MasterChef contract needs this access, the owner can assign roles at any time. AccessControl model makes the token vulnerable to malicious actions of the owner or role administrator.

Twin team response: issue will be fixed with a timelock contract.

Update: the ADMIN role of the TWIN contract was transferred to a Timelock contract (out of scope of current audit) in transactions with hashes 0x7fccda3037780ca55ff287844ebd5d137fbd3e0cf07d0827ceb934e533a26566, 0x6d22b7f4b2d2318cf63dd09b315679ee9f53b69be8dcd747ecbf85d16bff6110.

#02 TWIN: vesting not working High

Wrong `daysElapsed` calculation in `vestTeamTokens()` L39 based on timestamps instead of seconds. Vesting doesn't work (mints with rate 1000 times lower than expected), tests also fail. We recommend using a separate contract for vesting.

Twin team response: issue will be fixed with a new vesting contract.

Update: vesting is deployed to `0xfd6F40E643d7933D16435224C825E14bda427Ef7` and mint permissions were granted to this contract. It must be noted that the deployed contract mints additional tokens to the original vesting but with a correct rate.

#03 Minter: centralized price feed High

Minter contract uses the OracleManager (with multiple ChainLinkLikeOracles) to access the prices. It's not the aggregator and prices could be set instantly by the owner. If most of the oracles have the same owner, the system will be fully centralized.

Twin team response: we will move it to a multi sig account in the future or a governance contract.

#04 MasterChefV2: no cap for twinPerBlock — High

`updateEmissionRate()` function is used for updating the `twinPerBlock` parameter. Although it's the onlyOwner function, we recommend adding safety guards — capping the new value. If the owner's account gets compromised or the owner acts maliciously, the attacker can set an arbitrary big value for the `twinPerBlock` variable. In such a case the number of tokens till cap will be minted soon and the token price will drop. It should be noted that the risks of this issue are the same as #01 and regard only the case when the owner's account is not trusted or is not properly secured.

Update: the issue was fixed.

#05 Minter: division before multiplication — Medium

Division errors could be reduced by mitigating the multiplication over the result of division. Check functions `buyoutPosition()`, `withdrawCollateral()`, `mintPosition()`, `emergencyClaim()`, `buyoutPartialPosition()`.

Update: the issue was fixed.

#06 Minter: relying on ERC20 implementation — Medium

Requirements of returning the true boolean value of transfers rely on ERC20 [3] support of tAsset. While the TAsset.sol contract is a full ERC20 standard token, the owner of the OracleManager contract can add a different tAsset address of a partially implemented token with no returning value on transfers.

Update: the issue was fixed.

#07 MasterChefV2: minting capped token — Medium

TWIN token is capped, so once the cap is reached the MasterChefV2 contract will fail on minting, and reward withdrawals will be unavailable.

Commentary on the update: implemented cap for `twinPerBlock` variable makes it possible to mint the TWIN token to its own cap of 1.8e8 in less than 2 years.

#08 TFundOracle: for() loop gas problem Medium

`calculateData()` function uses `for()` loop with not limited max number of iterations and can possibly exceed block gas limit. `latestRoundData()` function calls `calculateData()` 3 times instead of once.

Update: the issue was fixed.

#09 TWIN: input data not checked Low

`changeTeamAddress()` and `vestTeamTokens()` functions lack checks on the input parameters.

#10 TWIN: variables naming Low

`teamMaxVestedTokens` should be written in UPPER_CASE format to conform to Solidity naming conventions [4] as it is a constant.

#11 Minter: unindexed events Low

Emitted events contain no indexed parameters which makes it harder to filter them.

Update: the issue was fixed.

#12 Minter: duplicated events Low

`Pause()` and `Unpause()` events duplicate the ones from Pausable.sol

Update: the issue was fixed.

#13 Minter: typos in error messages Low

Typos in error messages L154, 180, 288, 296, 332.

Update: the issue was fixed.

#14 Minter: input data not checked Low

Functions `updateBuyoutFeeAddress()`,`updateOracleAddress()`, `updateCollateralToken()` lack zero address checks on input values.

Update: the issue was fixed.

#15 Minter: CollateralToken gas optimization — Low

Structure `CollateralToken` could be optimized for gas savings via reordering or parameters. Should be {`uint256; address; bool`} or {`address; bool; uint256`}.

Update: the issue was fixed.

#16 Minter: variables naming — Low

`buyoutFeeUL` should be written in UPPER_CASE format to conform to the Solidity naming conventions.

Commentary on the update: content of this issue mistakenly was a copy of #11 issue in the first version of this document.

#17 Minter: TODO comment — Low

Code contains TODO comment in L321.

Update: the issue was fixed.

#18 CDPToken: unindexed events — Low

Emitted events contain no indexed parameters, which makes it harder to filter them.

Update: the issue was fixed.

#19 ChainLinkLikeOracle: only for decimals <= 18 — Low

OracleManager requires decimals of ChainLinkLikeOracle to be less or equal 18. There are no such checks nor comments on correct usage in the ChainLinkLikeOracle contract.

Update: the issue was fixed.

Commentary on the update: checks on correct decimals are performed in OracleManager.

#20 OracleManager: empty constructor — Low

The constructor function is empty and should be removed.

Update: the issue was fixed.

#21 OracleManager: typos in function name — Low

Typo in `getOracleMinimunCollateral()` function name L78.

Update: the issue was fixed.

#22 MaterChefV2: input data not checked — Low

Lack of checks in constructor input parameters L66-76.

#23 MaterChefV2: variables naming — Low

Violation of naming conventions [4] for variable Twin L43, should be in mixedCase.

#24 MaterChefV2: redundant code — Low

`BONUS_MULTIPLIER` constant is equal to 1 and could be removed.

Update: the issue was fixed.

#25 MaterChefV2: typos in comments — Low

Typo on “muliplier” L46.

Update: the issue was fixed.

#26 General recommendations — Low

We recommend using the fixed pragma version and naming the variables according to Solidity code style.

The audited code provided with tests, coverage is close to 100%. However, tests fail on TWIN token.

Functions lack documentation. We recommend adding NatSpec documentation at least for public and external functions and classes.

Contracts are extremely dependent on the owner’s account.

Conclusion

4 high severity issues were 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.

Audit includes recommendations on the code improving and preventing potential attacks.

Update: Twin Finance team has responded to this report. Most of the issues were fixed. 3 high severity issues remain in the updated code. It must be noted that 2 (#1 and #3) of these issues regard the case when the owner of the contract is not trusted or its account is not properly secured. To address #2 a vesting contract was deployed with a correct minting rate. Individual responses were added after each item in the section. Updated contracts are located in @twin-finance/contracts Github repository after the `6f61b0a` commit.

Updated contracts are deployed to BSC at:

TWIN	`0x62907AD5c2D79e2A4F048a90Ae2B49d062a773f3`,
Minter	`0x4cA7eA2a21230dF34B8b7C3576B0790aEa2F8877`,
MasterChefV2	`0x6287fFd0f2fFF3ac7Bc39751ed138188B83bB711`,
TFundOracle	`0x072C52006784163C406a0031051Bf8b5eE184383`,
TAsset	`0x79E29E397f33A4F515c11b913e5357327AE14aEC`,
	`0xc40bdAA708dA1446c13a82a5AFc18A396f40Bb40`,
	`0x7191d810103d34EEc28E04c61298551f2381fa6e`,
ChainLinkLikeOracle	`0xc6B22438f927b5C3bb70d138dABF6dcC590CaAdB`,
CDPToken	`0x1537a004D9f46a835Baf4a05c0dE1CbbB93e1B32`,
OracleManager	`0xa62DfAb32563c742FC61dd8B4c68095346DD3AA0`.

References

1. SushiSwap audit by PeckShield
2. SushiSwap audit by Quantstamp
3. ERC-20 standard
4. Solidity Docs: Naming Styles

Appendix A. Issues' severity classification

We consider an issue critical, if it may cause unlimited losses or breaks the workflow of the contract and could be easily triggered.

High severity issues may lead to limited losses or break interaction with users or other contracts under very specific conditions.

Medium severity issues do not cause the full loss of functionality but break the contract logic.

Low severity issues are typically nonoptimal code, unused variables, errors in messages. Usually, these issues do not need immediate reactions.

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