#HashEx
BLOCKCHAIN SECURITY

UniDex

smart contracts
final audit report

August 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 UniDex team to perform an audit of their smart contract. The audit was conducted between 11/08/2022 and 17/08/2022.

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 at Metis Network:

Router 0x1AA263d79E1f70409CE9159bb1A51F7844010a01,

Trading 0x7419694a09C21FC4D75f461d348520F3E8A506F6,

Oracle 0xCf0F646520b1e787b907dC2210f959E611849F72,

Pool 0x9Ba3db52BC401F4EF8ba23e56268C3AdE0290837,

Pool Rewards 0x615B1fcf461249b12342726819CB6Da23413CB48.

2.1 Summary

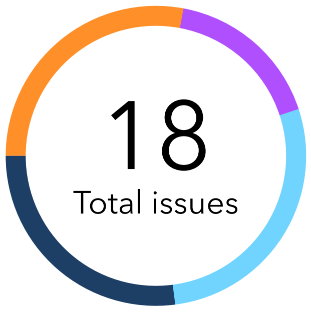
Project name	UniDex
URL	https://www.unidex.exchange/
Platform	Metis

Language	Solidity

2.2 Contracts

Name	Address
Oracle.sol	`0xCf0F646520b1e787b907dC2210f959E611849F72`
Pool.sol	`0x9Ba3db52BC401F4EF8ba23e56268C3AdE0290837`
PoolRewards.sol	`0x615B1fcf461249b12342726819CB6Da23413CB48`
Router.sol	`0x1AA263d79E1f70409CE9159bb1A51F7844010a01`
Trading.sol	`0x7419694a09C21FC4D75f461d348520F3E8A506F6`
All contracts	

3. Found issues



●	High	5 (28%)
●	Medium	3 (17%)
●	Low	5 (28%)
●	Info	5 (27%)

C1. Oracle.sol

ID	Severity	Title	Status
C1-01	● High	Excessive dark oracle's rights	✓ Acknowledged

C2. Pool.sol

ID	Severity	Title	Status
C2-01	● High	Excessive owner rights	✓ Acknowledged
C2-02	● Low	Discrepancies in utilization rate multiplier	✓ Acknowledged
C2-03	● Info	Depositors may loose their funds	✓ Acknowledged

C3. PoolRewards.sol

ID	Severity	Title	Status
C3-01	🟠 High	Excessive owner rights	✓ Acknowledged

C4. Router.sol

ID	Severity	Title	Status
C4-01	🟠 High	Excessive owner rights	✓ Acknowledged

C5. Trading.sol

ID	Severity	Title	Status
C5-01	🟠 High	Excessive owner rights	✓ Acknowledged
C5-02	🟣 Medium	Order's margin is not returned to a user in releaseMargin()	✓ Acknowledged
C5-03	🔵 Low	State change after external call	✓ Acknowledged
C5-04	🔵 Low	Gas optimization	✓ Acknowledged
C5-05	🔵 Low	Extra paid ETH is not returned	✓ Acknowledged
C5-06	● Info	Tokens with commissions are not supported	✓ Acknowledged
C5-07	● Info	Pool reserves may be insufficient to reward traders	✓ Acknowledged
C5-08	● Info	uint64 type can limit supported tokens range	✓ Acknowledged
C5-09	● Info	Discrepancies in fees structure on liquidations	✓ Acknowledged

C6. All contracts

ID	Severity	Title	Status
C6-01	● Medium	Lack of checks of the input values	✓ Acknowledged
C6-02	● Medium	No tests were provided	✓ Acknowledged
C6-03	● Low	Lack of events	✓ Acknowledged

4. Contracts

C1. Oracle.sol

Overview

Via this contract, the priviledged account `darkOracle` cans prices to the trading contract and liquidate positions in batches. Each `requestsPerFunding` request in a batch treasury sends `costPerRequest * requestsPerFunding` ETH to the dark oracle.

Issues

C1-01 Excessive dark oracle's rights ● High Acknowledged

The dark oracle has the right to `settleOrders()` and `liquidatePositions()` . If the dark oracle account is compromised, an attacker can set arbitrary prices for positions leading to liquidation or exaggerated profits for positions paid from the pools. Also, an attacker can withdraw all `treasury` funds spamming settle or liquidation batches where each `requestsPerFunding` request treasury will send him `costPerRequest * requestsPerFunding` ETH.

Recommendation

Transfer `darkOracle` ownership to multi signature wallet by Gnosis.

C2. Pool.sol

Overview

The contract aggregates staked funds for traders' rewards. Positions' margin losses are sent to the pool. Stakers get extra rewards proportionately to deposit from collected in trading contract fees.

Issues

C2-01 Excessive owner rights ● High ✓ Acknowledged

The contract owner has the right to change the `trading` address. If the owner account is compromised, an attacker can change the `trading` address to himself and withdraw all funds with `creditUserProfit()`.

Recommendation

Remove router change functionality and disable `trading` address change possibility in router or transfer `Pool` and `Router` ownership to `Timelock` contract with a minimum delay of at least 24 hours. This won't stop the admin and the owner from possible right abuses but it will help users to be informed about upcoming changes.

C2-02 Discrepancies in utilization rate multiplier ● Low ✓ Acknowledged

The utilization multiplier is set by default to 100.

```
    uint256 public utilizationMultiplier = 100; // in bps
```

In the function withdraw another hardcoded multiplier of value 10e4 is used:

```
  function withdraw(uint256 currencyAmount) external {
                ...
   uint256 utilization = getUtilization();
   require(utilization < 10**4, "!utilization");
                ...
                uint256 availableBalance = currentBalance * (10**4 - utilization) / 10**4;
   uint256 currencyAmountAfterFee = currencyAmount * (10**4 - withdrawFee) / 10**4;
                ...
        }
```

which supposedly leads to a wrong pool utilization calculation.

C2-03 Depositors may loose their funds ● Info ✓ Acknowledged

Staked on the contract currency is used for traders' rewards. A streak of successful trades or malicious owner behavior may lead to currency pool assets draining and partial or complete stakers funds loss.

Team response

This is nothing to different from other perp platform and the overall risk pooler take when becoming market makers (aka poolers) on the protocol. Its not exactly to different from market making at peak efficiency on CEXs and holding an undesirable trade (ex bitmex in 2020). Trading fees, funding rate, spread, and slippage all help counteract streak of successful wins and backstopping blackswan events.

C3. PoolRewards.sol

Overview

The contract holds the part of collected in trading contract fees and calculates rewards for currency pool stakers.

Issues

C3-01 Excessive owner rights ● High ✓ Acknowledged

The contract owner has the right to change the `treasury` address. If the owner account is compromised, an attacker can drain all funds changing `treasury` address to himself, flash loan pool, tremendously increase `pendingReward` with `notifyRewardReceived()` function, then update rewards and collect them.

Recommendation

Remove router change functionality and disable `treasury` address change possibility in router or transfer `PoolRewards` and `Router` ownership to `Timelock` contract with a minimum delay of at least 24 hours. This won't stop the admin and the owner from possible right abuses but it will help users to be informed about upcoming changes.

C4. Router.sol

Overview

The contact is addressed by other contracts for auxiliary mappings of currency to its decimals, pool contract, reward pool contract, rewards contract, and pools shares.

Issues

C4-01 Excessive owner rights ● High ✓ Acknowledged

The currencies decimals parameter is settable by the owner. If the owner account is compromised, an attacker can adjust currency value to extract assets from `Pool`, `PoolRewards`, `Treasury`, `Trading` contracts.

Recommendation

Remove decimals change functionality or transfer `Router` ownership to `Timelock` contract with a minimum delay of at least 24 hours. This won't stop the admin and the owner from possible right abuses but it will help users to be informed about upcoming changes.

C5. Trading.sol

Overview

Margin trading contract. A user buys currency trying to guess asset price change direction. The contract allows a user to increase the deposit by leveraging the loan. The user's position can be forcibly closed if provided margin is insufficient to cover losses.

Issues

C5-01 Excessive owner rights ● High ✓ Acknowledged

a. If the owner account is compromised, a malefactor can manipulate products fees to withdraw traders' deposits;

b. Prices on currencies are received from off-chained backend service and can be compromised to get extra earnings;

c. If the owner account is compromised, a malefactor can increase the product threshold and liquidate all positions.

C5-02 Order's margin is not returned to a user in releaseMargin() ● Medium ✓ Acknowledged

The `releaseMargin()` function returns to the user margin accumulated in his position. The function deletes both the active user's position and order, although the order's margin is not returned. If the user has opened an order during the `releaseMargin()` call, the margin of the unclosed order will be locked on the contract.

Recommendation

Return margin of open order if one exists.

C5-03 State change after external call ● Low ✓ Acknowledged

In `liquidatePosition()` function `position` is deleted after funds transfer. Although state change after an external call may be the reason for serious security leaks allowing reentrancy exploit, in this case, the method has limited access for `Oracle` contract and the issue can be abused only by the owner by liquidating same `position` multiple times.

C5-04 Gas optimization ● Low ✓ Acknowledged

a. `postition` should be placed in memory in L671,936;

b. `product` should be placed in memory in L889;

c. Excessive storage reads in L843,849,852-855,862.

C5-05 Extra paid ETH is not returned ● Low ✓ Acknowledged

Accidentally attached ETH to `submitOrder()` calls with a `currency` parameter not equal to `address(0)` is not returned to `msg.sender`. The issue is also relevant for assets received directly with `fallback()` and `receive()` functions. Such ETH will stay locked in the contract.

C5-06 Tokens with commissions are not supported ● Info ✓ Acknowledged

The contract logic doesn't consider that the actual received token amount can differentiate from the transferred. Adding tokens with a commission will inevitably lead to token exhaustment and gap between debt and the contract's balance.

Team response

Only the native gas token and USDC would ever be used by choice.

C5-07 Pool reserves may be insufficient to reward traders ● Info ✓ Acknowledged

Traders are sent profit from the currency pool. A streak of successful trades or malicious owner behavior may lead to currency pool assets draining and therefore cause funds shortage to pay a profit.

C5-08 uint64 type can limit supported tokens range ● Info ✓ Acknowledged

Tokens with `totalSupply` greater than `uint64.max` might be problematic for trading since max available for deposit amount will have insignificant value.

C5-09 Discrepancies in fees structure on liquidations ● Info ✓ Acknowledged

There are two ways of how an open position can be liquidated: when the oracle calls the `liquidatePosition()` function or when a user submits a close order request and liquidation requirements are met. In the first case fees for treasury are taken as `position.margin - threshold` whereas in the second case they are calculated as `order.size * product.fee / 10**6;`

Recommendation

Consider achieving fee consistancy for both liquidation cases by any appropriate way.

Team response

If a users is able to submit a close order request before keepers are able to process the liquidation then the remaining amount between liquidation threshold and the remaining margin can be closed and wouldnt be pocketed for profit on the protocol given keepers did not process that liquidation. This is not an error but rather just a user was closing a trade before able to be liquidatable by a keeper which can happen for a number of reasons. Lets say the user closes the trade at 79% drawn down before it hits the 80% threshold mark. The backend DON may now receive this order and assign a price which may leave it at a 85% negative draw down due to fast price movement. Thus this is treated as a standard closing order only fair to the user unless a keeper picked up this order as it reached the threshold and confirmed before the users close order.

C6. All contracts

Overview

Issues in this section are relevant for all contracts in the scope.

Issues

C6-01 Lack of checks of the input values ● Medium ✓ Acknowledged

There are no checks or constraints of input parameters validity in the contracts for ownable functions. For example, a fee equal to 100% or greater may be set in the Trading contract.

```
function updateProduct(bytes32 productId, Product memory _product) external onlyOwner {

 Product storage product = products[productId];

 require(product.liquidationThreshold > 0, "!product-does-not-exist");

 product.maxLeverage = _product.maxLeverage;
 product.fee = _product.fee;
 product.interest = _product.interest;
 product.liquidationThreshold = _product.liquidationThreshold;

}
```

Recommendation

As far as the contracts at the time of the audit are already deployed the project owner should update parameters with extra care.

C6-02 No tests were provided

● Medium ✓ Acknowledged

The project has complex, funds sensitive logic including trading math, fees charging, and integer calculations. We recommend adding tests with coverage of at least 90% to ensure precise work in any use case.

C6-03 Lack of events

● Low ✓ Acknowledged

The project incredibly lacks of events. We recommend emitting events on important value changes to be easily tracked off-chain. No events are emitted in important governance functions in every contract in the scope.

5. Conclusion

5 high, 3 medium, 5 low severity issues were found during the audit. No issues were resolved in the update.

The reviewed contract is extremely dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured as the users' funds and whole project operation are in control of the project's privileged accounts.

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