CERTIK

MDEX

Security Assessment

April 7th 2021

Summary

Overview

- Project Summary
- Engagement Summary
- Understandings
  - assets/Airdrop
  - assets/AirdropMDX
  - assets/BlackHole
  - assets/Repurchase
  - governance/GovernorAlpha
  - governance/Timelock
  - governance/TeamTimeLock
  - mainnet/CoinChef
  - mainnet/MdxToken
  - oracle/Oracle
  - heco

Findings

- CTK-MDEX#2-01 | Undeclared variable
- CTK-MDEX#2-02 | A typo in the oracle contract
- CTK-MDEX#2-03 | Incorrect contract addresses
- CTK-MDEX-01 | Dangerous Time-based Calculation
- CTK-MDEX-02 | Unreachable Function
- CTK-MDEX-03 | Wrong Assembly Code
- CTK-MDEX-04 | Missing override specifier
- CTK-MDEX-05 | Wrong Inheritance Hierarchy
- CTK-MDEX-06 | Wrong Constant Value
- CTK-MDEX-07 | Inconsistent Solidity Version
- CTK-MDEX-08 | Function Return Value Ignored
- CTK-MDEX-09 | Over Privileged Ownerships
- CTK-MDEX-10 | `minter()` Function Permission Not Restricted
- CTK-MDEX-11 | `add()` Function Input Not Restricted
- CTK-MDEX-12 | Checks Effects Interaction Pattern Not Used
- CTK-MDEX-13 | Proper Usage of `public` and `external`

Appendix | Finding Categories

Disclaimer

About CertiK

Summary

This report has been prepared for MDEX smart contracts, one the leading currency trading platforms on the heco chain, to discover issues and vulnerabilities in the source code as well as any dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing static analysis and manual review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by security experts.

The security assessment resulted in 13 findings that ranged from minor to informational. We recommend addressing these findings as potential improvements that can benefit the long run. We have done rounds of communications over the general understanding and the MDEX team has resolved the questions promptly.

Overall the source code is well written with security practices. The business logic is comprehensive and implemented accordingly. Yet we suggest a few recommendations that could better serve the project from the security perspective:

1. Enhance general coding practices for better structures of source codes;
2. Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
3. Provide more comments per each function for readability, especially contracts that are verified in public.
4. Provide more transparency on privileged activities once the protocol is live and gradually give the privileged permissions to the community once the protocol reaches a certain level of decentralization.

Overview

Project Summary

Name	MDEX
Codebase	https://github.com/mdexSwap/contracts/tree/master/contracts

Engagement Summary

Delivery Date	Mar 24th, 2021
Methodology	Static analysis, manual review and testnet simulation
Commit Hash	f9650a130c67c4b7804e8f92355ad7a7d2d50722; 02fd25468d0215ba0030c1713cc84e9a46cfb8c2

Understandings

MDEX is viewed as the composite DeFi ecosystem that integrates DEX, IMO, and DAO. It is one of the largest decentralized protocols deployed and maintained at HECO. Mdex is an automatic market-making decentralized exchange based on the concept of funding pools. Besides the same foundation as other DEX protocols, tt proposes and implements a dual-chain DEX model based on the Huobi Eco Chain and Ethereum .It combines the advantages of the low transaction fees of the Huobi Eco Chain and the prosperity of the Ethereum ecosystem, and supports the dual mining mechanism of liquidity and transactions.

assets/Airdrop

Airdrop is the standard vault/masterchef implementation where users deposit LP tokens in return for the rewards of wrapped HT. Enough checks applied to newAirdrop() by the owner to make sure the system parameters are aligned with enough coverage of WHT tokens.

assets/AirdropMDX

Airdrop is the standard vault/masterchef implementation where users deposit LP tokens in return for the rewards of MDEX's protocol native token. Enough checks applied to newAirdrop() by the owner to make sure the system parameters are aligned with enough coverage of MDX tokens.

assets/BlackHole

BlackHole is an empty smart contract that no one would be able to retrieve MDX from. It is used as the burner/destroyer address that associates with the Repurchase. Whenever the team repurchase MDX with USDT, and swap to location would be this BlackHole and the community can always check the balance of MDX that indicates the amount of the repurchase there:
https://hecoinfo.com/address/0xF9852C6588b70ad3c26daE47120f174527e03a25

assets/Repurchase

Repurchase is used by the MDEX team for the purpose of swapping USDT to MDEX then `burn` to a destroyer address. Owner could define the amount to be repurchased and a set of callers have the access to invoke the swap and complete the repurchase movement. Community could monitor the transactions of the smart contract, together with the asset holdings of BlackHole, to verify the e orts of the MDEX team on maintaining the financial factors of its protocol.

governance/GovernorAlpha

GovernorAlpha is the standard implementation of the governance system in light of the widely used open source version from compound.finance. A proposal created would be live voting for a period of time, then a succeeded one shall be queued into the timelock and get executed in the end to achieve the decentralized e orts of the community

governance/Timelock

The Timelock contract can modify system parameters, logic, and contracts in a 'time-delayed, opt-out' upgrade pattern. The Timelock has a hard-coded minimum delay of 2 days (maximum of 30 days), which is the least amount of notice possible for a governance action.

governance/TeamTimeLock

The TeamTimeLock is used for locking an amount of MDX tokens that are reserved for the team, and allows it to be withdrawn based on the time elapsed: every 30 days with 24 cycle times.

mainnet/CoinChef

The CoinChef is a standard implementation similar to terms like vault or masterchef, that allows end-users to stake LP tokens in return for token rewards. The innovation part introduced is the so-called double mining, where sushi pools are also supported that would allow end-users to stake both LP tokens on mdex and sushi and in return earning rewards of both MDX and SUSHI.

mainnet/MdxToken

The MdxToken is the standard implementation of ERC20 protocol with minimal extra features on top. It's the representation of MDX on Ethereum and it's mintable by privileged minter role accounts.

oracle/Oracle

The Oracle contract follows accurate time-weighted average prices (TWAPs) as the principle where could be referenced via sample oracle implementation there. It is used to view price information about a given asset from a given pair.

heco

The heco folder contains the core functionalities of MDEX's swapping protocol. In general:

1. Factory: The contract for creating pairs and includes utility functions to calculate the required token amount or the maximum output amount for a swap operation.
2. HecoPool: The vault contract that allows users to deposit LP tokens in return for MDX or other token as rewards. The "owner" can add new LPs to the pool to provide more options for users to earn rewards. The contract calculates and distributes rewards to the LP token providers when they call the deposit and withdraw function. Unlike most farming projects in the space, the HecoPool enables farming with the "multLp" token.
3. MdxTokenHeco: The token contract follows ERC20 with additional functionalities on voting and minting;

4. Router: The contract includes the handler for add and remove liquidity in a different scenario. It also includes a group of functions with the prefix "swap" in its function name to implement various ways to swap tokens. The main difference between various swap methods is they take a distinct path from the input token to the output token. ;
5. SwapMining: The core contract for swapping implementations that share the similar functionalities as UniSwap, with additional features that would allow liquidity providers to earn MDX during the swap procedure.

Findings

ID	Severity	Response
CTK-MDEX#2-01	minor	Pending
CTK-MDEX#2-02	minor	Resolved
CTK-MDEX#2-03	minor	Pending
CTK-MDEX-01	minor	Pending
CTK-MDEX-02	minor	Pending
CTK-MDEX-03	minor	Pending
CTK-MDEX-04	minor	Pending
CTK-MDEX-05	minor	Pending
CTK-MDEX-06	minor	Pending
CTK-MDEX-07	minor	Pending
CTK-MDEX-08	Informational	Pending
CTK-MDEX-09	Informational	Pending
CTK-MDEX-10	informational	Pending
CTK-MDEX-11	minor	Pending
CTK-MDEX-12	minor	Pending
CTK-MDEX-13	Informational	Pending

CTK-MDEX#2-01 | Undeclared variable

Type	Severity	Location
Volatile Code	Minor	bscContracts/mdxToken.sol: constructor()

Description

The variable `preMineSupply` is used in the `_mint()` function inside the contract constructor, but the variable is not defined in the contract.

Recommendation

Declare the `preMineSupply` variable and assign a value to it.

CTK-MDEX#2-02 | A typo in the oracle contract

Type	Severity	Location
Compiler Error	Minor	bscContracts/oracle.sol: consult()

Description

There is a letter `q` at the end of line 273, this causes an error when compiling the contract.

Recommendation

We assume this is a typo, the letter `q` should be removed.

Alleviation

The update has been applied here:
https://github.com/mdexSwap/contracts/commit/c9de3a25d4db6dc3e0c5231f4428b46232e104f1

CTK-MDEX#2-03 | Incorrect contract addresses

Type	Severity	Location
Volatile Code	Minor	bscContracts/repurchase.sol: L850-L853

Description

Multiple contracts' address is defined from line 850 to line 853 in the Repurchase contract. We find that those addresses belong to contracts deployed in the Heco chain instead of the Binance smart chain.

Recommendation

Update addresses to point them to specific contracts deployed in the Binance smart chain.

CTK-MDEX-01 | Dangerous Time-based Calculation

Type	Severity	Location
Volatile Code	minor	oracle/Oracle.sol: consult()

Description

In function `consult()`, `priceAverage` would be updated based on the variable `timeElapsed`. It is possible to call `update()` first and then immediately call `consult()`. In this scenario, the value of `timeElapsed` would be rather small, and thus `priceAverage` could get a relatively large value.

Recommendation

Recommend adding a `require()` check to make sure a minimal period of time between the calls of `update()` and `consult()`. Also, it is not recommended updating the value of `priceAverage` in `consult()`.

CTK-MDEX-02 | Unreachable Function

Type	Severity	Location
Volatile Code	Minor	governance/Timelock.sol: setPendingAdmin()

Description

Function `setPendingAdmin` has a `require` statement checking `msg.sender == address(this)`, which means `external` calls are forbidden. However, the function is never called in the codebase.

Recommendation

Recommend either modify the `require()` check or add a helper function in the contract to call the setter function.

CTK-MDEX-03 | Wrong Assembly Code

Type	Severity	Location
Compiler Error	Minor	heco/Factory.sol: constructor() of MdexERC20

Description

The local variable chainId is assigned to chainid, which is not a valid assembly code:

```
constructor() public {
   uint chainId;
   assembly {
       chainId := chainid
   }
   ...
}
```

Recommendation

Recommend correcting the assembly code to `chainId := chainid()`.

CTK-MDEX-04 | Missing override specifier

Type	Severity	Location
Compiler Error	Minor	heco/Factory.sol

Description

Based on the assumption that the contracts are to be deployed using 0.6.12, as the truffle-config file is using 0.6.12. There are multiple showings of functions lacking the `override` specifier. This will lead to failure of compiling the contracts with solidity 0.6.12.

Recommendation

For the inheriting functions, it is required to add `virtual` to every non-interface function intended to override, and to add `override` to the overriding functions, according to the Solidity 0.6.0 Breaking Changes.

CTK-MDEX-05 | Wrong Inheritance Hierarchy

Type	Severity	Location
Compiler Error	Minor	heco/Factory.sol: contract MdexPair

Description

Contract `MdexPair` is inherited from `MdexERC20` and `IMdexPair`.

1. There are compiler errors for `DOMAIN_SEPARATOR`, `PERMIT_TYPEHASH`, `allowance`, `balanceOf`, `decimals`, `name`, `nonces`, `symbol`, and `totalSupply`:

   ```
   Derived contract must override function "xxx". Two or more base
   classes define function with same name and parameter types. Since one
   of the bases defines a public state variable which cannot be
   overridden, you have to change the inheritance layout or the names of
   the functions.
   ```

2. There are compiler errors for `approve`, `permit`, `transfer` and `transferFrom`:

   ```
   Derived contract must override function "xxx". Two or more base
   classes define function with same name and parameter types.
   ```

CTK-MDEX-06 | Wrong Constant Value

Type	Severity	Location
Compiler Error	Minor	mainnet/CoinChef.sol

Description

`constant mdxPerBlock` is assigned to `100 ** 1e18`, which seems too large.

Recommendation

It seems the value should be `100 * 1e18` instead of `100 ** 1e18`.

CTK-MDEX-07 | Inconsistent Solidity Version

Type	Severity	Location
Compiler Error	Minor	heco/Router.sol oracle/Oracle.sol

Description

The contract versions of heco/Router.sol and oracle/Oracle are locked at 0.6.6, while truffle-config is using 0.6.12.

Recommendation

It is okay to try different compiler versions during the development stage.

However, we recommend locking the contract version when it reaches the production stage, and in this case seems 0.6.12 is more compatible.

CTK-MDEX-08 | Function Return Value Ignored

Type	Severity	Location
Volatile Code	Informational	CoinChef.sol: L85, L393, L395

Description

1. Return values of function `IERC20(_addLP).approve(sushiChef,uint256(- 1))` are ignored in function `addSushiLP(address)`.
2. Return values of `mdx.transfer(_to,mdxBal);mdx.transfer(_to,_amount)` are ignored in the function `safeMdxTransfer()`.

Recommendation

We advise developers to handle the return values of aforementioned functions to check if the transfer is executed without any error.

CTK-MDEX-09 | Over Privileged Ownerships

Type	Severity	Location
Control Flow	Informational	Airdrop, AirdropMDX, Repurchase, HecoPool, MdxTokenHeco, Router, SwapMining, CoinChef

Description

There are functions with modifier `onlyOwner` in the mentioned contracts, where the owner account could set some state variables, new airdrops, add new LP tokens, etc. These privileged behaviors could all potentially damage the economic system if abused without obtaining the consensus of the community.

Recommendation

Renounce ownership when it is the right timing; or gradually migrate to a timelock plus multisig governing procedure and let the community to monitor in respect of transparency considerations.

CTK-MDEX-10 | `minter()` Function Permission Not Restricted

Type	Severity	Location
Control Flow	informational	MdxToken: setMinter()

Description

In `MdxToken`, function `setMinter` has its scope declared as `external`, without any modifiers. Everyone could call `setMinter` and let their own address be the `minter` in this case. Since function `mint` is only restricted by checking if `minter` is equal to `msg.sender` in `onlyMinter`, with the unprotected function `setMinter`, everyone could `mint` unlimitedly.

Recommendation

Consider assigning the contract creator as the first minter and then transfer to a different one if necessary. We understand that the team had handled this well during the post deployment stage as if such a thing happens the MDX could be simply replaced with a newly created contract.

CTK-MDEX-11 | `add()` Function Input Not Restricted

Type	Severity	Location
Volatile Code	minor	CoinChef, Airdrop, AirdropMDX, HecoPool

Description

The comments mentioned `// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.`

The total amount of reward in function `updatePool()` will be incorrectly calculated if the same LP token is added into the pool more than once in function `add()`.

However, the code is not reflected in the comment behaviors as there isn't any valid restriction on preventing this issue.

The current implementation is relying on the trust of the owner to avoid repeatedly adding the same LP token to the pool, as the function will only be called by the owner.

Recommendation

Using mapping of `addresses -> booleans`, which can restrict the same address being added twice.

CTK-MDEX-12 | Checks Effects Interaction Pattern Not Used

Type	Severity	Location
Volatile Code	Minor	CoinChef, Airdrop, AirdropMDX, HecoPool

Description

In functions `deposit()` and `withdraw()` of the four contracts, the Checks Effects Interaction Pattern is not strictly followed. Using interfaces, the implementation of `safeTransfer` or `safeTransferFrom` are unknown and may have a malicious logical implementation that calls back to the function `deposit()`. This is dangerous to the calculation like the user's balance, the pool’s totalAmount, etc.

Recommendation

We advise developers to strictly follow the Checks-Effects-Interactions Pattern to avoid reentrancy and potential assets lost.

CTK-MDEX-13 | Proper Usage of `public` and `external`

Type	Severity	Location
Gas Optimization	Informational	General

Description

`public` functions that are never called by the contract could be declared `external`. When the inputs are arrays `external` functions are more efficient than `public` functions.

Recommendation

Consider using the `external` attribute for functions never called from the contract.

- newAirdrop(uint256,uint256,uint256): Airdrop.newAirdrop(uint256,uint256,uint256) (assets/Airdrop.sol#83-96)
- setCycle(uint256): Airdrop.setCycle(uint256) (assets/Airdrop.sol#106-108)
- add(uint256,IERC20,bool): Airdrop.add(uint256,IERC20,bool) (assets/Airdrop.sol#112-125)
- set(uint256,uint256,bool): Airdrop.set(uint256,uint256,bool) (assets/Airdrop.sol#128-134)
- deposit(uint256,uint256): Airdrop.deposit(uint256,uint256) (assets/Airdrop.sol#179-195)
- withdraw(uint256,uint256): Airdrop.withdraw(uint256,uint256) (assets/Airdrop.sol#198-213)
- emergencyWithdraw(uint256): Airdrop.emergencyWithdraw(uint256) (assets/Airdrop.sol#216-224)
- newAirdrop(uint256,uint256,uint256): AirdropMDX.newAirdrop(uint256,uint256,uint256) (assets/AirdropMDX.sol#79-92)
- setCycle(uint256): AirdropMDX.setCycle(uint256) (assets/AirdropMDX.sol#102-104)
- add(uint256,IERC20,bool): AirdropMDX.add(uint256,IERC20,bool) (assets/AirdropMDX.sol#108-122)
- set(uint256,uint256,bool): AirdropMDX.set(uint256,uint256,bool) (assets/AirdropMDX.sol#125-131)
- deposit(uint256,uint256): AirdropMDX.deposit(uint256,uint256) (assets/AirdropMDX.sol#186-205)
- withdraw(uint256,uint256): AirdropMDX.withdraw(uint256,uint256) (assets/AirdropMDX.sol#208-226)
- emergencyWithdraw(uint256): AirdropMDX.emergencyWithdraw(uint256) (assets/AirdropMDX.sol#229-240)
- setAmountIn(uint256): Repurchase.setAmountIn(uint256) (assets/Repurchase.sol#32-34)
- setEmergencyAddress(address): Repurchase.setEmergencyAddress(address) (assets/Repurchase.sol#36-39)
- addCaller(address): Repurchase.addCaller(address) (assets/Repurchase.sol#41-44)
- delCaller(address): Repurchase.delCaller(address) (assets/Repurchase.sol#46-49)
- getCaller(uint256): Repurchase.getCaller(uint256) (assets/Repurchase.sol#59-62)
- emergencyWithdraw(address): Repurchase.emergencyWithdraw(address) (assets/Repurchase.sol#78-81)
- propose(address[],uint256[],string[],bytes[],string): GovernorAlpha.propose(address[],uint256[],string[],bytes[],string) (governance/GovernorAlpha.sol#138-176)
- queue(uint256): GovernorAlpha.queue(uint256) (governance/GovernorAlpha.sol#178-187)
- execute(uint256): GovernorAlpha.execute(uint256) (governance/GovernorAlpha.sol#194-202)

- cancel(uint256): GovernorAlpha.cancel(uint256) (governance/GovernorAlpha.sol#204-217)
- getActions(uint256): GovernorAlpha.getActions(uint256) (governance/GovernorAlpha.sol#219-222)
- getReceipt(uint256,address): GovernorAlpha.getReceipt(uint256,address) (governance/GovernorAlpha.sol#224-226)
- castVote(uint256,bool): GovernorAlpha.castVote(uint256,bool) (governance/GovernorAlpha.sol#250-252)
- castVoteBySig(uint256,bool,uint8,bytes32,bytes32): GovernorAlpha.castVoteBySig(uint256,bool,uint8,bytes32,bytes32) (governance/GovernorAlpha.sol#254-261)
- __acceptAdmin(): GovernorAlpha.__acceptAdmin() (governance/GovernorAlpha.sol#283-286)
- __abdicate(): GovernorAlpha.__abdicate() (governance/GovernorAlpha.sol#288-291)
- __queueSetTimelockPendingAdmin(address,uint256): GovernorAlpha.__queueSetTimelockPendingAdmin(address,uint256) (governance/GovernorAlpha.sol#293-296)
- __executeSetTimelockPendingAdmin(address,uint256): GovernorAlpha.__executeSetTimelockPendingAdmin(address,uint256) (governance/GovernorAlpha.sol#298-301)
- mint(address,uint256): MdxToken.mint(address,uint256) (heco/MdxTokenHeco.sol#253-259)
- addMinter(address): MdxToken.addMinter(address) (heco/MdxTokenHeco.sol#261-264)
- delMinter(address): MdxToken.delMinter(address) (heco/MdxTokenHeco.sol#266-269)
- getMinter(uint256): MdxToken.getMinter(uint256) (heco/MdxTokenHeco.sol#279-282)
- setDelay(uint256): Timelock.setDelay(uint256) (governance/Timelock.sol#36-43)
- acceptAdmin(): Timelock.acceptAdmin() (governance/Timelock.sol#45-51)
- setPendingAdmin(address): Timelock.setPendingAdmin(address) (governance/Timelock.sol#53-58)
- queueTransaction(address,uint256,string,bytes,uint256): Timelock.queueTransaction(address,uint256,string,bytes,uint256) (governance/Timelock.sol#60-69)
- cancelTransaction(address,uint256,string,bytes,uint256): Timelock.cancelTransaction(address,uint256,string,bytes,uint256) (governance/Timelock.sol#71-78)
- executeTransaction(address,uint256,string,bytes,uint256): Timelock.executeTransaction(address,uint256,string,bytes,uint256) (governance/Timelock.sol#80-105)
- price(address,uint256): MdexPair.price(address,uint256) (heco/Factory.sol#343-352)
- quote(uint256,uint256,uint256): MdexFactory.quote(uint256,uint256,uint256) (heco/Factory.sol#438-442)
- getAmountsOut(uint256,address[]): MdexFactory.getAmountsOut(uint256,address[]) (heco/Factory.sol#464-472)
- getAmountsIn(uint256,address[]): MdexFactory.getAmountsIn(uint256,address[]) (heco/Factory.sol#475-483)
- setHalvingPeriod(uint256): HecoPool.setHalvingPeriod(uint256) (heco/HecoPool.sol#84-86)
- setMdxPerBlock(uint256): HecoPool.setMdxPerBlock(uint256) (heco/HecoPool.sol#89-92)
- addMultLP(address): HecoPool.addMultLP(address) (heco/HecoPool.sol#98-102)
- getMultLPAddress(uint256): HecoPool.getMultLPAddress(uint256) (heco/HecoPool.sol#112-115)
- setPause(): HecoPool.setPause() (heco/HecoPool.sol#117-119)
- setMultLP(address,address): HecoPool.setMultLP(address,address) (heco/HecoPool.sol#121-125)
- replaceMultLP(address,address): HecoPool.replaceMultLP(address,address) (heco/HecoPool.sol#127-140)
- add(uint256,IERC20,bool): HecoPool.add(uint256,IERC20,bool) (heco/HecoPool.sol#144-160)
- set(uint256,uint256,bool): HecoPool.set(uint256,uint256,bool) (heco/HecoPool.sol#163-169)
- setPoolCorr(uint256,uint256): HecoPool.setPoolCorr(uint256,uint256) (heco/HecoPool.sol#172-175)
- deposit(uint256,uint256): HecoPool.deposit(uint256,uint256) (heco/HecoPool.sol#300-307)

- withdraw(uint256,uint256): HecoPool.withdraw(uint256,uint256) (heco/HecoPool.sol#367-374)
- emergencyWithdraw(uint256): HecoPool.emergencyWithdraw(uint256) (heco/HecoPool.sol#422-429)
- addPair(uint256,address,bool): SwapMining.addPair(uint256,address,bool) (heco/SwapMining.sol#85-101)
- setPair(uint256,uint256,bool): SwapMining.setPair(uint256,uint256,bool) (heco/SwapMining.sol#104-110)
- setMdxPerBlock(uint256): SwapMining.setMdxPerBlock(uint256) (heco/SwapMining.sol#113-116)
- addWhitelist(address): SwapMining.addWhitelist(address) (heco/SwapMining.sol#119-122)
- delWhitelist(address): SwapMining.delWhitelist(address) (heco/SwapMining.sol#124-127)
- setHalvingPeriod(uint256): SwapMining.setHalvingPeriod(uint256) (heco/SwapMining.sol#142-144)
- setRouter(address): SwapMining.setRouter(address) (heco/SwapMining.sol#146-149)
- setOracle(IOracle): SwapMining.setOracle(IOracle) (heco/SwapMining.sol#151-154)
- phase(): SwapMining.phase() (heco/SwapMining.sol#167-169)
- reward(): SwapMining.reward() (heco/SwapMining.sol#176-178)
- swap(address,address,address,uint256): SwapMining.swap(address,address,address,uint256) (heco/SwapMining.sol#226-260)
- takerWithdraw(): SwapMining.takerWithdraw() (heco/SwapMining.sol#263-284)
- getUserReward(uint256): SwapMining.getUserReward(uint256) (heco/SwapMining.sol#287-299)
- getPoolInfo(uint256): SwapMining.getPoolInfo(uint256) (heco/SwapMining.sol#302-313)
- addSushiLP(address): CoinChef.addSushiLP(address) (mainnet/CoinChef.sol#83-87)
- getSushiLPAddress(uint256): CoinChef.getSushiLPAddress(uint256) (mainnet/CoinChef.sol#97-100)
- add(uint256,IERC20,bool): CoinChef.add(uint256,IERC20,bool) (mainnet/CoinChef.sol#104-120)
- set(uint256,uint256,bool): CoinChef.set(uint256,uint256,bool) (mainnet/CoinChef.sol#123-129)
- setPoolCorr(uint256,uint256): CoinChef.setPoolCorr(uint256,uint256) (mainnet/CoinChef.sol#132-135)
- deposit(uint256,uint256): CoinChef.deposit(uint256,uint256) (mainnet/CoinChef.sol#232-239)
- withdraw(uint256,uint256): CoinChef.withdraw(uint256,uint256) (mainnet/CoinChef.sol#299-306)
- emergencyWithdraw(uint256): CoinChef.emergencyWithdraw(uint256) (mainnet/CoinChef.sol#354-361)
- mint(address,uint256): MdxToken.mint(address,uint256) (mainnet/MdxToken.sol#10-13)
- getBalance(): TeamTimeLock.getBalance() (timeLock/TeamTimeLock.sol#44-46)
- setBeneficiary(address): TeamTimeLock.setBeneficiary(address) (timeLock/TeamTimeLock.sol#71-74)

Appendix | Finding Categories

Gas Optimization

Refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Refer to exhibits that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Refer to exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Business Model

Refer to contract or function logics that are debatable or not clearly implemented according to the design intentions.

Disclaimer

About CertiK

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

CERTIK

Provable Trust For All