Code Security Assessment

DeOrderBook

Jan 25th, 2022

Table of Contents

Summary

This report has been prepared for DeOrderBook to discover issues and vulnerabilities in the source code of the DeOrderBook project as well as any contract 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 industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	DeOrderBook
Platform	other
Language	Solidity
Codebase	https://github.com/DeOrderBook/v1-core
Commit	6f4c56b32988ddb5ad5d9c7b93e1089e2634367b

Audit Summary

Delivery Date	Jan 25, 2022
Audit Methodology	Static Analysis, Manual Review

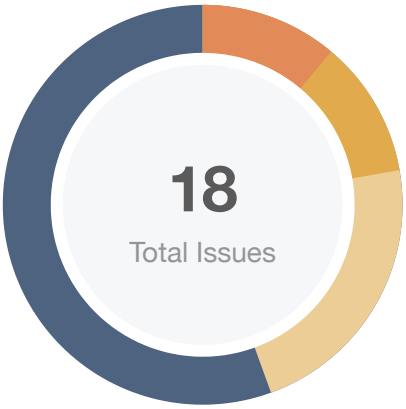
Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Mitigated	Resolved
Critical	0	0	0	0	0	0	0
Major	2	0	0	2	0	0	0
Medium	2	0	0	0	0	0	2
Minor	4	0	0	2	1	0	1
Informational	10	0	0	7	0	0	3
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
GCC	governance/GovernanceControl.sol	3bf993958c4ea558b933a58f994246fb49e33516ed02f87a38660f8edaf88e32
GBD	governance/GovernorBravoDelegate.sol	da2f76c7c573b1a2d558d6a3914efaa4507ce9dc3a51dd1a8b7d09bfab765362
GBC	governance/GovernorBravoDelegator.sol	109a00acd7aeeffbb956e59003917acd6a90624e55800cd274d3b321ada0b341
TLC	governance/TimeLock.sol	7a961ba94df324f4fc0d997b34d9bf9ffa80eef83174ba9d0f833a89e5deda5a
TCK	governance/Treasury.sol	4fb0ef8a8329c3ea918572023ab7a8a47ee669cee05f7aa544e6daa5e7acb6a3
DCK	option/Distributions.sol	19ccb9a95a5a05a53dd303c9ac05e42c57332f16de88be59a375a2a675331ddc
OCK	option/Option.sol	1a59cd2b2778b8d4f8f9e9f20c992045e494dd88e1ec7c5e836e9e807aadb932
OFC	option/OptionFactory.sol	49444c573933bd99584b2af3ef9f1d0f5e00d2aac8c39700d5b274da46fde15b
DOB	staking/DOBStakingPool.sol	339ea7f2b747fa020315670bfde31bcd8846675b268b31236e8b45031eb50da3
SPR	staking/StakingPoolRewarder.sol	3116575c6c72156c82b01eba5a8933947395af6fa5201fa1c4e18925a3e7ac3d
SPC	staking/StakingPools.sol	f0bec4d7fb6231c403318687570ec72c497bedd3aaa8500471e2da4363e51f16
BCK	token/Bullet.sol	2c9fae4d453d3813cb5dcba75743ab1ff7c31466cd8c8ece048e9e00a010691f
DOT	token/DOBToken.sol	184fc0ee78baa8d426a871377932bc1466872fa81f28c39155320d7c0a10399f
HOD	token/HODLToken.sol	375a67f362b26e18f6851983945ec9bd9040afc4578d3caed568c8bbf0a455c2
SCK	token/Sniper.sol	2b0e20e2a3bd66a6dbc1aa7f54eb4d42dc06fa74fd9055035c46fcbc91296bb5

Findings



Critical	**0** (0.00%)
Major	**2** (11.11%)
Medium	**2** (11.11%)
Minor	**4** (22.22%)
Informational	**10** (55.56%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Acknowledged
GLOBAL-02	Division Before Multiplication	Mathematical Operations	Minor	Acknowledged
GLOBAL-03	Redundant SafeMath in Solidity 0.8.x	Language Specific	Informational	Acknowledged
DCK-01	`delete` Without Status Checks	Volatile Code	Informational	Acknowledged
DOB-01	Missing Pause Limitation on `redeemReward()`	Inconsistency, Control Flow	Informational	Resolved
DOB-02	Unable to Unpause	Control Flow	Informational	Acknowledged
DOT-01	Centralized Holding Positions	**Centralization / Privilege**	**Major**	Acknowledged
GBD-01	Inconsistency between Code and Comments for `MIN_VOTING_PERIOD`	Inconsistency	Informational	Resolved
GBD-02	Zero Value for `MIN_VOTING_DELAY`	Volatile Code	Informational	Acknowledged
GBD-03	Proposals Cannot Be Rejected before Initiating as Governor	Volatile Code	Minor	Resolved
HOD-01	Potential Precision Loss	Mathematical Operations	Minor	Acknowledged

ID	Title	Category	Severity	Status
OCK-01	`initialize()` Can Be Called Multiple Times in Option	Control Flow	Medium	Resolved
OCK-02	Discontinuous Time Periods	Volatile Code	Minor	Partially Resolved
OCK-03	Uninitialized Local Variable	Volatile Code	Informational	Acknowledged
OFC-01	Uninitialized State Variables	Volatile Code	Informational	Resolved
SPC-01	Missing Restrictions for Unstaking	Volatile Code	Informational	Acknowledged
STA-01	Checks-Effects-Interactions Pattern Not Applied	Volatile Code, Coding Style	Informational	Acknowledged
TOK-01	`initialize()` Can Be Called Multiple Times in Sniper and Bullet	Control Flow	Medium	Resolved

GLOBAL-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	Major	Global	Acknowledged

Description

In the contract `GovernanceControl`, the role `onlyGovernance` has authority over the following functions:

- setGovernance()
- setExecutor()

In the contract `GovernorBravoDelegate`, the role `admin` has authority over the following functions:

- initialize()
- setVotingDelay()
- setVotingPeriod()
- setProposalThreshold()
- setWhitelistAccountExpiration()
- setWhitelistGuardian()
- initiate()
- setPendingAdmin()

In the contract `GovernorBravoDelegate`, the role `whitelistGuardian` has authority over the following functions:

- setWhitelistAccountExpiration()

In the contract `GovernorBravoDelegator`, the role `admin` has authority over the following functions:

- setImplementation()

In the contract `Treasury`, the role `onlyGovernance` has authority over the following functions:

- increaseAllowance()
- decreaseAllowance()
- transfer()

In the contract `Distributions`, the role `onlyOwner` has authority over the following functions:

- setExerciseFee()

- setWithdrawFee()
- setHodlWithdrawFee()
- setBulletToRewardRatio()
- setFeeDistribution()
- setBulletDistribution()
- setHodlWithdrawFeeDistribution()

In the contract `Option`, the role `onlyFactory` has authority over the following functions:

- setup()

In the contract `OptionFactory`, the role `onlyOwner` has authority over the following functions:

- setStakingPools()
- setDOBStakingPool()
- setDistributions()
- setStakingRewardPerBlock()
- notifyRemoveBullet()

In the contract `DOBStakingPool`, the role `onlyWorker` has authority over the following functions:

- drawReward()

In the contract `DOBStakingPool`, the role `onlyFactory` has authority over the following functions:

- addBullet()
- removeBullet()

In the contract `DOBStakingPool`, the role `onlyOwner` has authority over the following functions:

- setWorker()
- setFactory()
- setuHODLRewarder()
- setbHODLRewarder()
- setBulletRewardThreshold()
- setExtendLockDays()
- pause()

In the contract `StakingPoolRewarder`, the role `onlyOwner` has authority over the following functions:

- updateVestingSetting()
- setRewardDispatcher()

In the contract `StakingPoolRewarder`, the role `onlyStakingPools` has authority over the following functions:

- onReward()
- claimVestedReward()

In the contract `StakingPools`, the role `onlyOwnerOrFactory` has authority over the following functions:

- createPool()

In the contract `Bullet`, the role `onlyController` has authority over the following functions:

- mintFor()
- burn()
- burnFrom()

In the contract `DOB`, the role `onlyAdmin` has authority over the following functions:

- setPendingAdmin()
- snapshot()

In the contract `Sniper`, the role `onlyController` has authority over the following functions:

- mintFor()
- burn()
- burnFrom()

Any compromise to the admin/privileged account may allow a hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles. OR
- Remove the risky functionality.

Alleviation

According to DeOrderBooks's feedabck, there are Timelock and DAO in place and will renounce the ownership to DAO.

GLOBAL-02 | Division Before Multiplication

Category	Severity	Location	Status
Mathematical Operations	Minor	Global	Acknowledged

Description

There are mathematical operations performing divisions before multiplications, which is possible to lead to precision loss:

- Option.enter()
- Option.exercise()
- Option.withdrawTarget()
- StakingPoolRewarder._calculateWithdrawableFromVesting()
- StakingPoolRewarder._calculateUnvestedAmountAtCurrentStep()
- StakingPoolRewarder._calculateUnvestedAmount()
- StakingPoolRewarder._onReward()
- HODLToken.withdrawTo()

Recommendation

Recommend performing multiplications before divisions to prevent precision loss, reviewing the decimals for the big numbers/multipliers/etc., and adding more tests to cover the potential precision loss.

Alleviation

[DeOrderBook]:

This is to prevent overflow. Tests performed comprehensively and mostly calculating precentage, not dividing both large decimal numbers. The precision loss can be beared

GLOBAL-03 | Redundant SafeMath In Solidity 0.8.x

Category	Severity	Location	Status
Language Specific	Informational	Global	Acknowledged

Description

In contracts Timelock, Option, OptionFactory, DOBStakingPool, StakingPoolRewarder, StakingPools, DOB and HODLToken, the SafeMath libraries are imported and used for math operations. This is redundant as in solidity 0.8.x, arithmetic operations will revert if overflow/underflow occurs.

Recommendation

Recommend removing the redundant import and adjust the codes.

DCK-01 | `delete` Without Status Checks

Category	Severity	Location	Status
Volatile Code	Informational	option/Distributions.sol: 77~78, 95~96, 113~114	Acknowledged

Description

In contract Distributions, public state variables `feeDistribution`, `bulletDistribution`, and `hodlWithdrawFeeDistribution` can respectively be `delete` and reset in functions `setFeeDistribution()`, `setBulletDistribution()`, and `setHodlWithdrawFeeDistribution()`.

These three setter functions have input validation on `_percentage.length == _to.length` and `sum == 100`. However, since these three public variables have dependencies on contracts Option and HODLToken, the `delete` and reset behavior would lead to data missing issues on the dependency function calls.

Our understanding is that those state variables are not meant to be changed during the period of `inExerciseTime` and `beforeExerciseTime`. Please let us know if our understanding is wrong.

Recommendation

Recommend adding checks to make sure it is safe to `delete` and reset the public state variables.

Alleviation

[DeOrderBook]:

Those are onlyOwner setter. There is logic to ensure the new distribution to replace with the old one.

Those variables are internally fee distribution and should be able to change even during the period of inExerciseTime and beforeExerciseTime according to our requirement.

DOB-01 | Missing Pause Limitation On `redeemReward()`

Category	Severity	Location	Status
Inconsistency, Control Flow	Informational	staking/DOBStakingPool.sol: 325~326	Resolved

Description

Unlike `stake()` and `unstake()`, there are no time period limitations on function `redeemReward()`. The only requirements to call `redeemReward()` are `nonReentrant` and `uHODLClaimable > 0 || bHODLClaimable > 0`. Thus `redeemReward()` can be freely called even if the whole contract is paused.

Recommendation

Recommend keeping consistency on the pausing conditions on the business logic functions.

Alleviation

Fixed in commit d79b4796d123b9bf9bad02a409abc01beb3a27da.

DOB-02 | Unable To Unpause

Category	Severity	Location	Status
Control Flow	Informational	staking/DOBStakingPool.sol: 364~365	Acknowledged

Description

According to the codes in the contract `DOBStakingPool`, the function `pause()` exists in the contract. However, the contract lacks the corresponding function `unpause()`.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

[DeOrderBook]:

The contract will be voided once paused and offer emergencyUnstake function to stakers

DOT-01 | Centralized Holding Positions

Category	Severity	Location	Status
Centralization / Privilege	**Major**	token/DOBToken.sol: 72~73	ⓘ Acknowledged

Description

The `MAX_SUPPLY` of the DOB tokens are sent to the contract deployer, `admin`, in the constructor function. This could be a centralization risk as the deployer can distribute DOB tokens without obtaining the consensus of the community.

Recommendation

Recommend making sure the transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

Alleviation

[DeOrderBook]:

Tokens will be allocated to different treasury after deployed and renouce the ownership to DAO

GBD-01 | Inconsistency Between Code And Comments For `MIN_VOTING_PERIOD`

Category	Severity	Location	Status
Inconsistency	Informational	governance/GovernorBravoDelegate.sol: 17~18	Resolved

Description

In contract GovernorBravoDelegate, there are lines of codes initializing constant state variables:

```
/// The minimum setable voting period
uint256 public constant MIN_VOTING_PERIOD = 1; // About 24 hours

/// The max setable voting period
uint256 public constant MAX_VOTING_PERIOD = 80640; // About 2 weeks
```

It seems that the value for `MIN_VOTING_PERIOD` should be 24 hours instead of 1. See L20, if `80640 == 2 weeks`, then 24 hours should be `5760`.

Also, the value seems not to be simple timestamp based on seconds, since `80640/14/24/60 = 4`.

Recommendation

Recommend properly documenting the constant values and make sure the code and comments are consistent and meet the design.

Alleviation

Fixed in commit 776da1d4bea26dc345ce9bde3d8e0a6c69f6de51.

GBD-02 | Zero Value For `MIN_VOTING_DELAY`

Category	Severity	Location	Status
Volatile Code	Informational	governance/GovernorBravoDelegate.sol: 23	Acknowledged

Description

In contract GovernorBravoDelegate, the constant state variable `MIN_VOTING_DELAY` is initialized as `0.

Recommendation

We guess the `0` value is for testing purpose, recommend assigning a valid value when project is about to launch.

Alleviation

[DeOrderBook]:

Will assgined valid value at launch.

GBD-03 | Proposals Cannot Be Rejected Before Initiating As Governor

Category	Severity	Location	Status
Volatile Code	Minor	governance/GovernorBravoDelegate.sol: 363~364	Resolved

Description

We noticed that the contract GovernorBravoDelegate is derived from the Compound's GovernorBravoDelegate. In Compound's version, there is a state variable `initialProposeId` to receive the proposal id from GovernorAlpha and also make sure function `purpose()` can only be called when the initiation is done. This variable is removed in the current version, thus `propose()` function cannot reject proposals when the initiation is not done.

Recommendation

We understand that the functionality of receiving proposal id is not needed, but still recommend add some similar checks to make sure the `propose()` function can only be called after the initiation.

Alleviation

Fixed in commit 776da1d4bea26dc345ce9bde3d8e0a6c69f6de51.

HOD-01 | Potential Precision Loss

Category	Severity	Location	Status
Mathematical Operations	Minor	token/HODLToken.sol: 49~50, 51~52	Acknowledged

Description

In function `withdrawTo()` of HODLToken.sol, the summation of `feeAmount / 100 * percentage` is not guaranteed to be equal to `feeAmount`. If `feeAmount` calculated in L45 is less than the summation of `feeAmount / 100 * percentage`, then the last transfer in L51 would revert; otherwise, if `feeAmount` is greater than the summation, then the last transfer in L51 would withdraw an amount which is slightly less than the `amount` given as the input parameter.

The similar situations also happen in Option.sol, function `enter()` may have precision loss on `bulletToReward`, function `exercise()` may have precision loss on `exerciseFee`, and function `withdrawTarget` may have precision loss on `withdrawFee`.

Recommendation

Recommend calculating the summation in this function and compare the delta between the summation and `feeAmount` in L45. Also, performing multiplication before division can help to minimize precision loss (please see GLOBAL-02).

Alleviation

[DeOrderBook]:

This is to prevent overflow. Tests performed comprehensively and it is for calculating precentage, not dividing both large decimal numbers. The precision loss can be beared

OCK-01 | `initialize()` Can Be Called Multiple Times In Option

Category	Severity	Location	Status
Control Flow	Medium	option/Option.sol: 65~66	Resolved

Description

In contract Option, the only check that the function `initialize()` has is to make sure that input `_type` parameter is less than or equal to `1`. However, there are no checks preventing that the `initialize()` function can be called multiple times.

Recommendation

Recommend adding some modifiers or checks with the `require` statement, as well as an appropriate error messages, so that the `initialize()` function can only be called once.

Alleviation

Fixed in commit 28545dcb915a8cc3d83dc5423d5143286575e5ae.

OCK-02 | Discontinuous Time Periods

Category	Severity	Location	Status
Volatile Code	Minor	option/Option.sol: 42~43, 47~48, 55~56	Partially Resolved

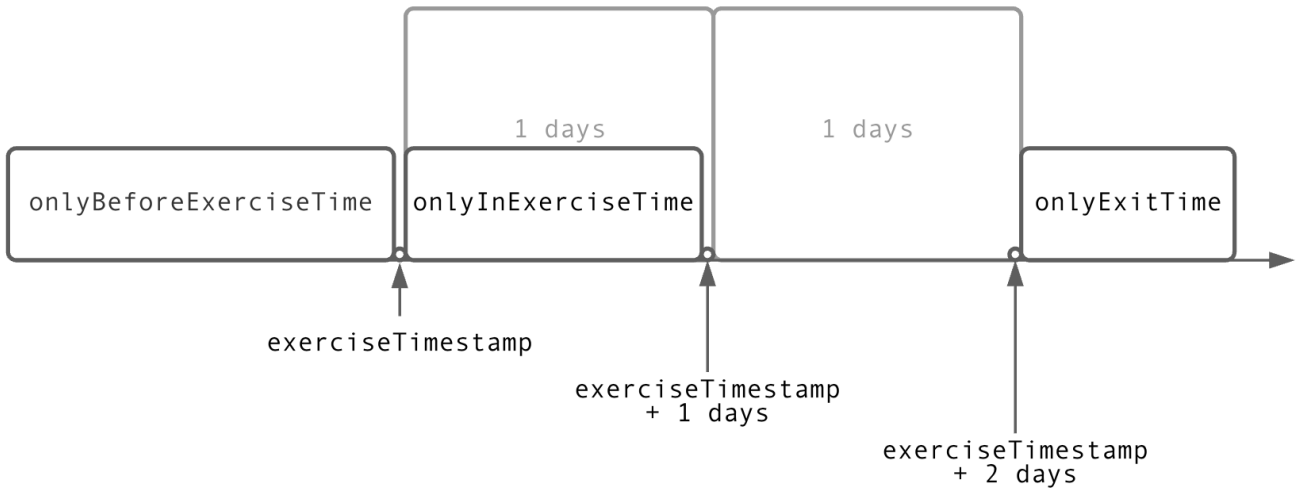
Description

There are three modifiers in contract Option. However, the time periods are not continuous. There is an `1 second` gap between `onlyBeforeExerciseTime` and `onlyInExerciseTime`, and an `1 day` gap between `onlyInExerciseTime` and `onlyExitTime`.

```
modifier onlyBeforeExerciseTime() {
  require(block.timestamp < exerciseTimestamp, "Option: only before exercise time");
  _;
}

modifier onlyInExerciseTime() {
  require(
      exerciseTimestamp < block.timestamp && block.timestamp < exerciseTimestamp + 1
days,
      "Option: only in exercise time"
  );
  _;
}

modifier onlyExitTime() {
  require(block.timestamp > exerciseTimestamp + 2 days, "Option: only in exit time");
  _;
}
```



Recommendation

Recommend making sure the gaps are desired and properly documenting the gaps in design documentation.

Alleviation

Partially fixed in commit 5a8af33b6ddddbd6683e580a64b57e08aca26451.

[DeOrderBook]:

This is the business requirement, not an issue, where after 1 day, the right of execrise expires.

OCK-03 | Uninitialized Local Variable

Category	Severity	Location	Status
Volatile Code	Informational	option/Option.sol: 135~136	Acknowledged

Description

Value of `baseAmount` would never be assigned if `optionType` is neither 0 or 1.

Recommendation

Recommend initializing the local variable to an acceptable default value.

Alleviation

[DeOrderBook]:

Option is created and initialized by OptionalFactory, there is check on _optionType in OptionFactory.createOption, the type can only be 0 or 1.

OFC-01 | Uninitialized State Variables

Category	Severity	Location	Status
Volatile Code	Informational	option/OptionFactory.sol: 26~27	Resolved

Description

Variable `fundOptions` is used in function `getOptionIDs()`, but it is never initialized nor used in other functions. It seems `fundOptions` is an unused variable and can be removed.

Recommendation

Recommend updating the variable implementation to meet the design specification

Alleviation

Fixed in commit c97dcd359a00b138b417d26027e9d9c32f82f2d3.

SPC-01 | Missing Restrictions For Unstaking

Category	Severity	Location	Status
Volatile Code	Informational	staking/StakingPools.sol: 220~221	Acknowledged

Description

In contract StakingPools, there are external functions `stake()` and `unstake()` for users to manage their staking. There are checks to make sure the `amount` is greater than 0 on these three functions, and there is a check in `stakeFor()` to make sure the `user` is not zero address.

However, it seems the restrictions on a minimal time period of staking is missing, which means users can stake and unstake in a very short period of time the arbitrage some rewards.

Our understanding is that those state variables are not meant to be changed during the period of `inExerciseTime` and `beforeExerciseTime`. Please let us know if our understanding is wrong.

Recommendation

Recommend adding some state variables and require statements to make sure the users cannot stake and unstake in an undesired minimal time period.

Alleviation

[DeOrderBook]:

Rewards is based on block and stake amount with vesting, The gas is more than the staker arbitrage from a very short period. Therefore it is not neccessary to put the restrictions.

Funds reserved in the staking pool is meant to rewards to stakers each blocks.

STA-01 | Checks-Effects-Interactions Pattern Not Applied

Category	Severity	Location	Status
Volatile Code, Coding Style	Informational	staking/DOBStakingPool.sol: 218~219 staking/StakingPoolRewarder.sol: 244~245, 274~275, 320~321	Acknowledged

Description

State variables are changed after function calls to external contract like `safeTransferFrom()`, `onReward()`, etc.

Recommendation

Recommend applying the Checks-Effects-Interactions Pattern for cases like this.

It shields public functions from re-entrancy attacks. It's always a good practice to follow this pattern. `checks-effects-interaction` pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.

Alleviation

[DeOrderBook]:

There are reentrancy lock in StakingPool and StakingPoolRewarder

TOK-01 | `initialize()` Can Be Called Multiple Times In Sniper And Bullet

Category	Severity	Location	Status
Control Flow	Medium	token/Sniper.sol: 22~23 token/Bullet.sol: 22~23	Resolved

Description

In contracts Sniper and Bullet, the only check that the function `initialize()` has is to make sure that input `_controller` parameter is not a zero address. However, there are no checks preventing that the `initialize()` function can be called multiple times.

Recommendation

Recommend adding some modifiers or checks with the `require` statement, as well as an appropriate error messages, so that the `initialize()` function can only be called once.

Alleviation

Fixed in commit ed905ed10881d5203c4912ff8fa8a6ae52f6219c.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Control Flow

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

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

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

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings 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.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

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.