DODO

Security Assessment

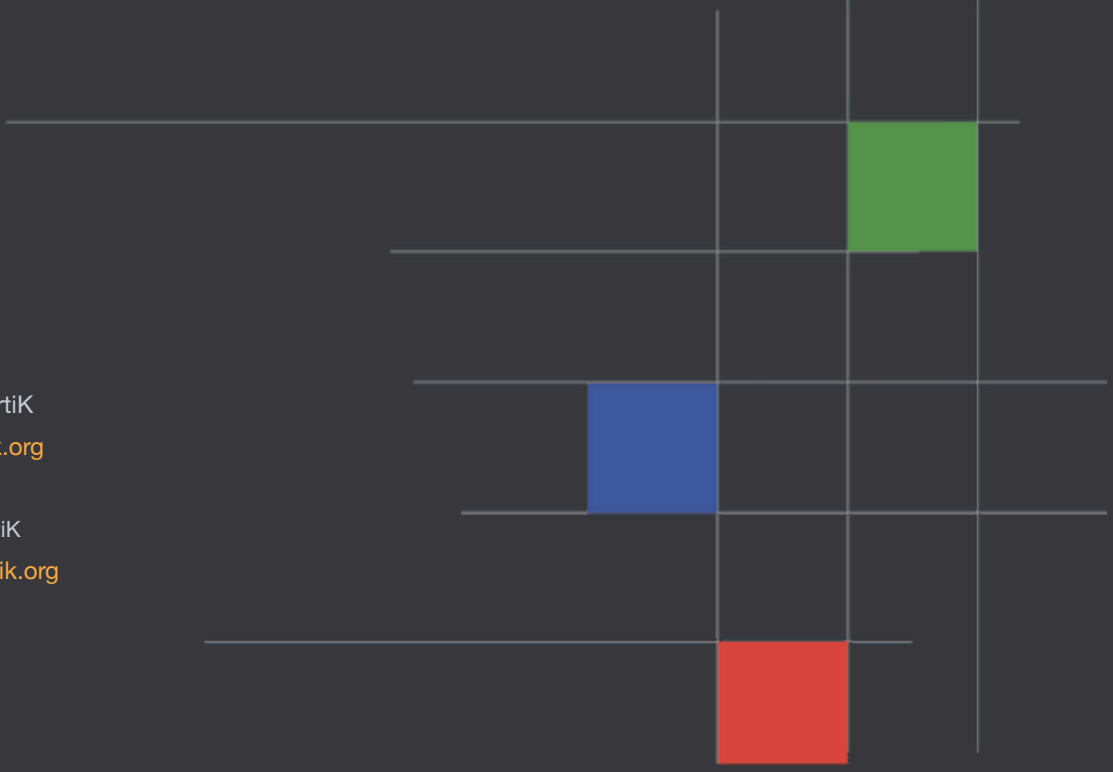
February 18th, 2021

For :
DODO

By :

Wythe Li @ CertiK
weizhi.li@certik.org

Bryan Xu @ CertiK
buyun.xu@certik.org



Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	DODO
Description	A next-generation on-chain liquidity provider
Platform	Ethereum; Solidity; Yul
Codebase	GitHub Repository
Commit	66de802e594ac307464215af115dee4fa45abd26

Audit Summary

Delivery Date	Feb. 18th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Feb. 14, 2021 - Feb. 18, 2021

Vulnerability Summary

Total Issues	11
Total Critical	0
Total Major	0
Total Minor	4
Total Informational	7

Executive Summary

This report has been prepared for **DODO** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, 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.

Additionally, to bridge the trust gap between administrator and users, administrator needs to express a sincere attitude with the consideration of the administrator team's anonymousness. The administrator has the responsibility to notify users with the following capability of the administrator:

- Owner can transfer assets in vDODOToken contract to owner via 'emergencyWithdraw' method

To improve the trustworthiness of the project, any dynamic runtime changes on the protocol should be notified to clients. Any plan to call these methods is better to move to the execution queue of Timelock, and also emit events.
The team confirmed that Timelock（registered in DODOApproveProxy) has been deployed, address is :
https://etherscan.io/address/0x335ac99bb3e51bdbf22025f092ebc1cf2c5cc619

File in Scope

ID	Contract	SHA-256 Checksum	Commit
VDT	vDODOToken.sol	e972a4fa1c2a34715d4d17734232a0657e3bcd3d3b431bd7edc7588da7e15c6e	66de802e594ac307464215af115dee4fa45abd26

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the **DODO** team or reported an issue.

Review Notes

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 4 minor vulnerabilities were identified during our audit that solely concerns the specification.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project has adequate doumentation and specification outside of the source files, also the code comment coverage is detailed.

Findings

ID	Title	Type	Severity
VDT-01	Incorrect Naming Convention Utilization	Coding Style	Informational
VDT-02	State variables that could be declared constant	Coding Style	Informational
VDT-03	Proper Usage of “public” and “external” type	Gas Optimization	Informational
VDT-04	Missing Emit Events	Optimization	Informational
VDT-05	Missing Checks	Logical Issue	Informational
VDT-06	`Checks-effects-pattern` Not Used	Implemention	Minor
VDT-07	Logic issue in `mint()`	Logical Issue	Informational
VDT-08	Missing Return Value Check for Transfer	Logical Issue	Minor
VDT-09	Variable Name *DOOD_GOV*	Coding Style	Informational
VDT-10	Owner initialize	Logical Issue	Minor
VDT-11	Type Casting	Implemention	Minor

VDT-01: Incorrect Naming Convention Utilization

Type	Severity	Location
Coding Style	Informational	vDODOToken.sol L36-60

Description:

Solidity defines a naming convention that should be followed. In general, parameters should use mixedCase, refer to: https://solidity.readthedocs.io/en/v0.6.12/style-guide.html#naming-conventions

Variables shoud use mixedCase.

Examples:
Variables like: `_ALLOWED_` , `_DODO_TOKEN_` , `_DODO_APPROVE_PROXY_` , `_DODO_TEAM_`

Functions other than constructors should use mixedCase.

Examples:
Functions like: `setCantransfer()`

Recommendation:

The recommendations outlined here are intended to improve the readability, and thus they are not rules, but rather guidelines to try and help convey the most information through the names of things.

VDT-02: State variables that could be declared constant

Type	Severity	Location
Coding Style	Informational	vDODOToken.sol L33-35

Description:

Below varaibles are not changed after initialization.

If they are constants, better to define as constants. Constant state variables should be declared constant to save gas.

```
        string public name = "vDODO Membership Token";
        string public symbol = "vDODO";
        uint8 public decimals = 18;
```

Recommendation:

We recommend to change the codes like below:

```
        string constant public name = "vDODO Membership Token";
        string constant public symbol = "vDODO";
        uint8 constant public decimals = 18;
```

Alleviation:

These two variables were removed in commit 77ba0bd903e3392dda67c2bd0b132e47b53b6ffe.

VDT-03: Proper Usage of "public" and "external" type

Type	Severity	Location
Gas Optimization	Informational	vDODOToken.sol

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.

Examples:

Functions `setCantransfer()` , `changePerReward()` , `updateDODOFeeBurnRatio()` , `updateDODOCirculationHelper()` , `updateGovernance()` , `emergencyWithdraw()` , `mint()` , `redeem()` , `donate()` , `preDepositedBlockReward()` , `totalSupply()` , `transfer()` , `approve()` , `transferFrom()` , `allowance()` , `getDODOWithdrawFeeRatio()` , `getSuperior()`

Recommendation:

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

VDT-04: Missing Emit Events

Type	Severity	Location
Optimization	Informational	vDODOToken.sol

Description:

Several sensitive actions are defined without event declarations.

Examples:
Functions like : `constructor()` , `emergencyWithdraw()` , `updateGovernance()` .

Recommendation:

Consider adding events for sensitive actions, and emit it in the function like below.

```
        event UpdateGovernance(address governance);
        function updateGovernance(address governance) public onlyOwner {
            _DOOD_GOV_ = governance;
            emit UpdateGovernance(governance);
        }
```

VDT-05: Missing Checks

Type	Severity	Location
Logical Issue	Informational	vDODOToken.sol

Description:

Functions `updateDODOCirculationHelper()`, `updateGovernance()`, `constructor()`, `redeem()`, `donate()`, `preDepositedBlockReward()` are missing checkings for parameters.

Recommendation:

We recommend to add neccessary checks, for example:

```
        function updateDODOCirculationHelper(address helper) public onlyOwner {
            require(helper != address(0), "Invalid circulation helper");
            _DODO_CIRCULATION_HELPER_ = helper;
        }
```

VDT-06: `Checks-effects-pattern` Not Used

Type	Severity	Location
Implementation	Minor	vDODOToken.sol

Description:

There are state variables changed after transfers are done in the functions `mint()` , `donate()` , and `preDepositedBlockReward()` of the contract. This may lead to reentrancy issue.

Recommendation:

It is recommended to follow checks-effects-interactions pattern. It shields public functions from re-entrancy attacks, refer to: https://docs.soliditylang.org/en/v0.8.0/security-considerations.html#re-entrancy

Alleviation:

The team confirmed that `mint()` 、 `donate()` 、 `preDepositedBlockReward()` only involve a same external contract call, and the contract is fixed as DODOApproveProxy(0x335aC99bb3E51BDbF22025f092Ebc1Cf2c5cC619
).

And this contract is open source, there is no reentrant logic.

VDT-07: Logic issue in `mint()`

Type	Severity	Location
Logical Issue	Informational	vDODOToken.sol L148-160

Description:

When the user already has a superior, the main logic of the function does not mention the parameter `superiorAddress`.

Recommendation:

Consider moving the requirement

```
            require(
                superiorAddress != address(0) && superiorAddress != msg.sender,
                "vDODOToken: Superior INVALID"   //
            );
```

into

```
            if (user.superior == address(0)) {
                require(
                    superiorAddress == _DODO_TEAM_ || userInfo[superiorAddress].superior !=
    address(0),
                    "vDODOToken: INVALID_SUPERIOR_ADDRESS"
                );
                // move here
                user.superior = superiorAddress;
            }
```

VDT-08: Missing Return Value Check for Transfer

Type	Severity	Location
Logical Issue	Minor	vDODOToken.sol

Description:

Missing return vallue check for transfers.

Examples:

Function `redeem` :

```
    IERC20(_DODO_TOKEN_).transfer(msg.sender, dodoReceive);
    IERC20(_DODO_TOKEN_).transfer(address(0), burnDodoAmount);
```

Function `emergencyWithdraw` :

```
    ERC20(_DODO_TOKEN_).transfer(_OWNER_, dodoBalance);
```

Recommendation:

We recommend to use `safeTransfer` .

```
        using SafeERC20 for IERC20;
        IERC20(_DODO_TOKEN_).safeTransfer(msg.sender, dodoReceive);
```

Alleviation:

The team confirmed that the transfer operation only invloves dodo erc20 Token （0x43dfc4159d86f3a37a5a4b3d4580b888ad7d4ddd）.And the token complies with erc20 standard specification.

Here is the source code of `dodoToken` :

```
        function transfer(address to, uint256 amount) public returns (bool) {
            require(amount <= balances[msg.sender], "BALANCE_NOT_ENOUGH");

            balances[msg.sender] = balances[msg.sender].sub(amount);
            balances[to] = balances[to].add(amount);
            emit Transfer(msg.sender, to, amount);
            return true;
        }
```

VDT-09: Variable Name `_DOOD_GOV_`

Type	Severity	Location
Coding Style	Informational	vDODOToken.sol

Description:

Variable name `_DOOD_GOV_` could be miswritten.

Recommendation:

Consider replacing it with `_DODO_GOV_` .

VDT-10: Owner initialize

Type	Severity	Location
Logical Issue	Minor	vDODOToken.sol

Description:

The contract implements `InitializableOwnable` interface, which means anyone first call the function `initOwner` becomes the owner of the contract `vDODOToken` .

It is possible that someone can seize the control of `vDODOToken`

Recommendation:

Consider initialize `owner` in constructor.

Alleviation:

The team confirmed that `vDODOToken` have been deployed on the mainnet (0xc4436fBAE6eBa5d95bf7d53Ae515F8A707Bd402A) and init the owner correctly.

VDT-11: Type Casting

Type	Severity	Location
Implemention	Minor	vDODOToken.sol

Description:

Type casting from `uint256` into `uint112` or `uint128` needs caution.

Recommendation:

Consider checking the validity of the number before type casting.

Example:

Function `_mint` :

```
        require(uint256(to.stakingPower).add(stakingPower) <= uint128(-1), "OVERFLOW");
        require(uint256(to.superiorSP).add(superiorIncreSP) <= uint128(-1), "OVERFLOW");
        to.stakingPower = uint128(uint256(to.stakingPower).add(stakingPower));
        to.superiorSP = uint128(uint256(to.superiorSP).add(superiorIncreSP));
```

Whenever you want to perform a type conversion, check the number first.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings 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

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

Logical Issue

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

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.

Data Flow

Data Flow findings 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

Coding Style findings 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

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.

Magic Numbers

Magic Number findings 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

Compiler Error findings 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.

Icons explanation

 : Issue resolved

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.