Smart Contract Security Audit Report

Contents

1. Executive Summary

On Mar. 15, 2021, the SlowMist security team received the DODO team's security audit application for DODOV2, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerabilities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.

Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack

- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background

3.1 Project Introduction

DODO is a decentralized exchange platform powered by the Proactive Market Maker (PMM) algorithm. It features highly capital-efficient liquidity pools that support single-token provision, reduce impermanent loss, and minimize slippage for traders. The trading platform also offers SmartTrade, a decentralized liquidity aggregation service that routes to and compares various liquidity sources to quote the optimal prices between any two tokens. In addition, DODO removed all roadblocks hindering liquidity pool creation for the issuance of new assets - asset ratios, liquidity depths, fee rates, and other parameters can all be freely customized and configured in real-time. Based on this breakthrough, DODO has developed Crowdpooling, a permissionless, equal opportunity liquidity offering mechanic, as well as customizable technical solutions geared towards professional on-chain market makers.

Audit Files

Contract source code:

Initial audit version:

Github: https://github.com/DODOEX/contractV2

Branch: main

commit: abe9919a01d17f8acc29fd0a2a95ad3c1bd49264

Branch: featureV2/mineV2AndDSP

commit: 3177452358c22f0d2ef6e4187da8ff0805970e81

Final audit version:

Github: https://github.com/DODOEX/contractV2

Branch: main

commit: 8da3ee1ec50966fca9a2c80d424040c45c0f785e

Audit scope:

Branch: main

```
contracts
├── CrowdPooling
├── DODOFee
├── DODOPrivatePool
├── DODOStablePool
├── DODOToken
├── DODOVendingMachine
├── Factory
├── SmartRoute
├── external
```

```
├── helper
├── intf
└── lib
```

4. Code Overview

4.1 Contract Visibility Analysis

During the audit, the SlowMist Security Team analyzed the visibility of the core contract. For details, please refer to the attachment *DODO_Visibility_Check.zip*

4.2 Code Audit

4.2.1 Low-risk vulnerabilities

4.3.1.1. The problem of DODOApprove pre-initialization

DODOApproveProxy has a 3-day grace period when switching a new DODOApprove contract. You need to wait for the new DODOApprove to take effect after the end of the buffer period. However, when the unlockAddProxy function is used to add the DODOApprove that needs to be switched, the DODOApprove contract is not initialized in time. As a result, malicious attackers can monitor the unlockAddProxy function and initialize the DODOApprove contract first.

```
    function unlockAddProxy(address newDodoProxy) public onlyOwner {
        _TIMELOCK_ = block.timestamp + _TIMELOCK_DURATION_;
        _PENDING_ADD_DODO_PROXY_ = newDodoProxy;
    }
```

Fix stauts: Ignored. After communicating with the project party, it is confirmed that the current

DODOApprove contract will not be upgraded for the time being, so ignore the problem.

4.3.1.2 Risk of loss of user funds

In the LockTokenVault contract, when transferring the user's locked token, it did not check whether the _to address is msg.sender itself, which caused the user to abuse the transfer and cause a loss of funds.

```
function transferLockedToken(address to) external {
        originBalances[to] = originBalances[to].add(originBalances[msg.sender]);
        claimedBalances[to] = claimedBalances[to].add(claimedBalances[msg.sender]);

        originBalances[msg.sender] = 0;
        claimedBalances[msg.sender] = 0;
    }
```

Fix status: fixed, repair commit: main-08a06609604779c31db493bc0d755efa1c3f0a61.

4.2.2 Enhancement Suggestions

4.2.2.1 Missing events

Contract： DODOApproveProxy ， Function list： init / unlockAddProxy / addDODOProxy / LockAddProxy / removeDODOProxy

Contract：DODOApprove，Function list：init / unlockSetProxy / setDODOProxy / LockSetProxy

Contract：DODV2Proxy02，Function list：addWhiteList / removeWhiteList / updateGasReturn

Contract：DVMFactor，Function list：updateDvmTemplate

Contract：DPPAdvance，Function list：tunePrice

Contract：DPPAdvance ，Function list：setOperator / setFreezeTimestamp

Contract：DPPVault，Function list：ratioSync / retrieve

Contract：DVMVault，Function list：_setReserve，_sync

The above functions does not have an event declaration, it is recommended to add the corresponding event declaration

Fix status: After communicating with the project party, it is confirmed that the above event statement is not currently used in business and will be fixed in subsequent iterations.

4.2.2.2 The contract balance was not verified when the reward was distributed

In the RewardVault contract, the contract balance is not verified when the reward is distributed, which may cause the contract balance to fail to be distributed

```
function reward(address to, uint256 amount) external onlyOwner {
    //SlowMist// Not verify if contract balance is larger than the transfer amount
    IERC20(dodoToken).safeTransfer(to, amount);
}
```

Fix situation: After confirming with the project party, they ignore this problem.

4.2.2.3 Unchecked array length

a. The getPendingReward function in the BaseMine contract did not verify whether the value of i passed in was less than the length of the array when obtaining the reward of the pool, which resulted in the failure to obtain the reward.

```
function getPendingReward(address user, uint256 i) public view returns (uint256) {
        //SlowMist// Not verify the array length
        RewardTokenInfo storage rt = rewardTokenInfos[i];
        uint256 accRewardPerShare = rt.accRewardPerShare;
        if (rt.lastRewardBlock != block.number) {
            accRewardPerShare = _getAccRewardPerShare(i);
        }
        return
```

```
            DecimalMath.mulFloor(
                balanceOf(user),
                accRewardPerShare.sub(rt.userRewardPerSharePaid[user])
            ).add(rt.userRewards[user]);
    }
```

Fix status: fixed, repair commit: feature/mineV2AndDSP- 5917c95439cac06241f108038043d22348c863e6.

b. The claimReward function in the BaseMine contract does not verify whether the value of i passed in is less than the length of the array, resulting in failure to obtain rewards

```
    function claimReward(uint256 i) public {
        require(i<rewardTokenInfos.length, "DODOMineV2: REWARD_ID_NOT_FOUND");
        _updateReward(msg.sender, i);
        RewardTokenInfo storage rt = rewardTokenInfos[i];
        uint256 reward = rt.userRewards[msg.sender];
        if (reward > 0) {
            rt.userRewards[msg.sender] = 0;
            IRewardVault(rt.rewardVault).reward(msg.sender, reward);
            emit Claim(i, msg.sender, reward);
        }
    }
```

Fix status: fixed, repair commit: feature/mineV2AndDSP- 5917c95439cac06241f108038043d22348c863e6.

4.2.2.4 Compatibility risk of rebasing tokens

The deposit function of the ERC20Mine contract does not verify the incoming amount. When it is compatible with rebasing tokens, it will cause an error to obtain the transfer amount.

```
    function claimReward(uint256 i) public {
        require(i<rewardTokenInfos.length, "DODOMineV2: REWARD_ID_NOT_FOUND");
        _updateReward(msg.sender, i);
```

```
        RewardTokenInfo storage rt = rewardTokenInfos[i];
        uint256 reward = rt.userRewards[msg.sender];
        if (reward > 0) {
            rt.userRewards[msg.sender] = 0;
            IRewardVault(rt.rewardVault).reward(msg.sender, reward);
            emit Claim(i, msg.sender, reward);
        }
    }
```

Fix status: fixed, fix commit: main-d26b21bd814d4bfcc702521d52f6cb3af4f86e5c.

5. Audit Result

5.1 Conclusion

Audit Result : Passed

Audit Number : 0X002104090002

Audit Date : April 09, 2021

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist Team in-house analysis tool audit of the codes for security issues. There are five security issues found during the audit. There are two low-risk vulnerabilities and three enhancement suggestion. After communication and feedback with the dFuture team, it was confirmed that the risks found in the audit process were repaired or within a tolerable range.

6. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist