Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.09.19, the SlowMist security team received the team's security audit application for JOJO Exchange, 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.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules 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.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using 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:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit
		Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

Audit Version:

Project address:

https://github.com/JOJOexchange/smart-contract-EVM

commit: 23403f169a903c8c238ff34803807ac178c660cc

Fixed Version:

Project address:

https://github.com/JOJOexchange/smart-contract-EVM

commit: 18e4f2a1e6790bdd8d9a799848f811bdf2860f65

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Missing event record	Others	Suggestion	Fixed
N2	Risk of excessive authority	Authority Control Vulnerability	Low	Fixed
N3	Risk of precision calculation	Arithmetic Accuracy Deviation Vulnerability	Low	Ignored

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

JOJODealer			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	JOJOStorage
version	External	-	-

JOJOExternal			
Function Name	Visibility	Mutability	Modifiers
deposit	External	Can Modify State	nonReentrant
requestWithdraw	External	Can Modify State	nonReentrant
executeWithdraw	External	Can Modify State	nonReentrant
isSafe	External	-	-
isAllSafe	External	-	-
getFundingRate	External	-	-
setOperator	External	Can Modify State	-
handleBadDebt	External	Can Modify State	-
requestLiquidation	External	Can Modify State	onlyRegisteredPerp
openPosition	External	Can Modify State	onlyRegisteredPerp

JOJOExternal			
realizePnl	External	Can Modify State	onlyRegisteredPerp
approveTrade	External	Can Modify State	onlyRegisteredPerp

JOJOStorage			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable

JOJOView			
Function Name	Visibility	Mutability	Modifiers
getRiskParams	External	-	-
getAllRegisteredPerps	External	-	-
getMarkPrice	External	-	-
getPositions	External	-	-
getCreditOf	External	-	-
isOrderSenderValid	External	-	-
isOperatorValid	External	-	-
getTraderRisk	External	-	-
getLiquidationPrice	External	-	-
getLiquidationCost	External	-	-
getOrderFilledAmount	External	-	-

JOJOOperation			
Function Name	Visibility	Mutability	Modifiers
updateFundingRate	External	Can Modify State	onlyFundingRateKeeper
setPerpRiskParams	External	Can Modify State	onlyOwner
setFundingRateKeeper	External	Can Modify State	onlyOwner
setInsurance	External	Can Modify State	onlyOwner
setWithdrawTimeLock	External	Can Modify State	onlyOwner
setOrderSender	External	Can Modify State	onlyOwner
setSecondaryAsset	External	Can Modify State	onlyOwner

Perpetual			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
balanceOf	External	-	-
updateFundingRate	External	Can Modify State	onlyOwner
getFundingRate	External	-	-
trade	External	Can Modify State	-
liquidate	External	Can Modify State	-
_settle	Internal	Can Modify State	-

Subaccount			
Function Name	Visibility	Mutability	Modifiers

Subaccount			
init	External	Can Modify State	-
setOperator	External	Can Modify State	onlyOwner
requestWithdraw	External	Can Modify State	onlyOwner
executeWithdraw	External	Can Modify State	onlyOwner
retrieve	External	Can Modify State	onlyOwner

SubaccountFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
newSubaccount	External	Can Modify State	-
getSubaccounts	External	-	-
getSubaccount	External	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Missing event record

Category: Others

Content

Modifying sensitive parameters in the contract lacks corresponding event records, which is not conducive to the supervision of the community and users.

Code location: contracts/lib/Operation.sol #L132-141

```
function setSecondaryAsset(
    Types.State storage state,
    address _secondaryAsset
) external {
    require(
        state.secondaryAsset == address(0),
        Errors.SECONDARY_ASSET_ALREASY_EXIST
    );
    state.secondaryAsset = _secondaryAsset;
}
```

Solution

It is recommended to add corresponding event records.

Status

Fixed

[N2] [Low] Risk of excessive authority

Category: Authority Control Vulnerability

Content

The FundingRateKeeper role can arbitrarily modify the funding rate of the contract by calling the updateFundingRate function. This will have a direct impact on perpetual contracts.

Code location: contracts/impl/JOJOOperation.sol#L24-28

```
function updateFundingRate(
    address[] calldata perpList,
    int256[] calldata rateList
) external onlyFundingRateKeeper {
    Operation.updateFundingRate(perpList, rateList);
}
```

The owner role can arbitrarily modify the risk parameters of the perpetual markets, set the time interval for withdrawal execution and set the secondary asset. This will have a direct impact on perpetual contracts and users' funds.

Code location: contracts/impl/JOJOOperation.sol

```
    function setPerpRiskParams(address perp, Types.RiskParams calldata param)
        external
        onlyOwner
    {
        Operation.setPerpRiskParams(state, perp, param);
    }
    ...
    function setWithdrawTimeLock(uint256 newWithdrawTimeLock)
        external
        onlyOwner
    {
        Operation.setWithdrawTimeLock(state, newWithdrawTimeLock);
    }
    ...
    function setSecondaryAsset(address _secondaryAsset) external onlyOwner {
        Operation.setSecondaryAsset(state, _secondaryAsset);
    }
```

The orderSender role can match transactions by calling the trader function and constructing any trader data. If the role has the risk of doing evil, it will affect normal transactions and user funds.

Code location: contracts/impl/Perpetual.sol

```
    function trade(bytes calldata tradeData) external {
        (
            address[] memory traderList,
            int256[] memory paperChangeList,
            int256[] memory creditChangeList
        ) = IDealer(owner()).approveTrade(msg.sender, tradeData);

        for (uint256 i = 0; i < traderList.length; ) {
            _settle(traderList[i], paperChangeList[i], creditChangeList[i]);
            unchecked {
                ++i;
            }
        }
        require(IDealer(owner()).isAllSafe(traderList), "TRADER_NOT_SAFE");
    }
```

Solution

It is recommended to transfer the authority of roles with excessive authorization risk to the governance contract, at least using multi-signature wallets

Status

Fixed; The project team response: We will use a 2 of 3 multi-signer to manage the owner, orderSender we will manage centrally, similar to the CEX hot wallet. fundingRateKeeper will be registered as the FundingRateUpdateLimiter contract, and the rate of change can only be limited by updating the fundingRate through the FundingRateUpdateLimiter contract.

[N3] [Low] Risk of precision calculation

Category: Arithmetic Accuracy Deviation Vulnerability

Content

When judging whether the account can be liquidated, because the paperAmount parameter is controlled by the order sender when constructing the transaction data, if the paperAmount parameter is set to a very small value, the maintenanceMargin will be equal to 0 after calculation, thus passing the judgment. This results in very small positions that can be opened without deposit at the time of trading and will not be liquidated.

Code location: contracts/lib/Liquidation.sol

```
function getTotalExposure(Types.State storage state, address trader)
    public
    view
    returns (
        int256 netPositionValue,
        uint256 exposure,
        uint256 maintenanceMargin
    )
{
        ...
```

```
         netPositionValue += paperAmount.decimalMul(price) + creditAmount;
         uint256 exposureIncrement = paperAmount.decimalMul(price).abs();
         exposure += exposureIncrement;
         maintenanceMargin +=
             (exposureIncrement * params.liquidationThreshold) /
             10**18;
         ...
     }
     ...
function _isAllSafe(Types.State storage state, address[] memory traderList)
     internal
     view
     returns (bool)
 {
         ...
             maintenanceMargin +=
                 (paperAmount.decimalMul(markPrice).abs() *
                     params.liquidationThreshold) /
                 10**18;
             netValue += paperAmount.decimalMul(markPrice) + credit;
             unchecked {
                 ++j;
             }
         }

         // return false if any one of traders is lack of collateral
         if (netValue < int256(maintenanceMargin)) {
             return false;
         }

         unchecked {
             ++i;
         }
     }
     return true;
 }
```

Solution

It is recommended to add a range limit to paperAmount parameter.

Status

Ignored; The project team response: We believe that firstly orderSender is credible, secondly the decimal of paper is 18, the amount of paper needed to produce the accuracy error is too small, it also can't cover the gas cost and is not profitable. Finally, we decided not to make the modification.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002209300002	SlowMist Security Team	2022.09.19 - 2022.09.30	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 low risk, 1 suggestion vulnerabilities. And 1 low risk was ignored; All other findings were fixed. The code was not deployed to the mainnet.

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 based 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 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.

SLOWMIST

Official Website

www.slowmist.com

✉

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist