Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.04.18, the SlowMist security team received the JOJO Exchange 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

Project address:

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

commit:

eddec47e74a9273f738eada74313d5964c4b2f86

Module:

impl + intf + lib + subAccount + utils

review version:

f137bb70f83c68fd6106c6aceb4e914f019d8ce8

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Lack of judgment on the balance of withdrawal users	Others	Suggestion	Ignored
N2	Order validation can be bypassed	Others	Low	Fixed
N3	Risk of excessive authority	Authority Control Vulnerability	Low	Ignored
N4	The perp address is not verified	Others	Suggestion	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	Ownable

JOJOExternal			
Function Name	Visibility	Mutability	Modifiers
deposit	External	Can Modify State	nonReentrant
requestWithdraw	External	Can Modify State	nonReentrant
executeWithdraw	External	Can Modify State	nonReentrant
approveTrade	External	Can Modify State	-
requestLiquidate	External	Can Modify State	-
positionClear	External	Can Modify State	-

JOJOOperation			
Function Name	Visibility	Mutability	Modifiers
handleBadDebt	External	Can Modify State	onlyOwner
updateFundingRate	External	Can Modify State	onlyOwner

JOJOOperation			
setPerpRiskParams	External	Can Modify State	onlyOwner
setInsurance	External	Can Modify State	onlyOwner
setWithdrawTimeLock	External	Can Modify State	onlyOwner
setSecondaryAsset	External	Can Modify State	onlyOwner

JOJOView			
Function Name	Visibility	Mutability	Modifiers
getPrimaryAsset	External	-	-
getSecondaryAsset	External	-	-
getRiskParams	External	-	-
getFundingRate	External	-	-
getRegisteredPerp	External	-	-
getPositions	External	-	-
getCreditOf	External	-	-
isSafe	External	-	-
isPositionSafe	External	-	-
getTraderRisk	External	-	-
getLiquidationPrice	External	-	-
getLiquidationCost	External	-	-
getOrderHash	External	-	-

JOJOView			
getOrderFilledAmount	External	-	-

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

Subaccount			
Function Name	Visibility	Mutability	Modifiers
init	External	Can Modify State	-
isValidPerpetualOperator	External	-	-
setOperator	External	Can Modify State	onlyOwner
requestWithdraw	External	Can Modify State	onlyOwner
executeWithdraw	External	Can Modify State	onlyOwner

SubaccountFactory			
Function Name	Visibility	Mutability	Modifiers

SubaccountFactory			
<Constructor>	Public	Can Modify State	-
newSubaccount	External	Can Modify State	-
getSubaccounts	External	-	-

EIP712			
Function Name	Visibility	Mutability	Modifiers
_buildDomainSeparator	Public	-	-
_hashTypedDataV4	Public	-	-

Funding			
Function Name	Visibility	Mutability	Modifiers
_deposit	Public	Can Modify State	-
_requestWithdraw	Public	Can Modify State	-
_executeWithdraw	Public	Can Modify State	-
_withdraw	Private	Can Modify State	-

Liquidation			
Function Name	Visibility	Mutability	Modifiers
_getLiquidationPrice	Public	-	-
_getTotalExposure	Public	-	-
_isSolidSafe	Public	-	-

Liquidation			
_isSafe	Public	-	-
_isPositionSafe	Public	-	-
_getLiquidateCreditAmount	External	-	-
_positionClear	External	Can Modify State	-

Trading			
Function Name	Visibility	Mutability	Modifiers
_approveTrade	Public	Can Modify State	-
_addPosition	Private	Can Modify State	-
_priceMatchCheck	Private	-	-
_structHash	Public	-	-
_getOrderHash	Public	-	-
_validateOrder	Public	Can Modify State	-
_info2MakerFeeRate	Private	-	-
_info2TakerFeeRate	Private	-	-
_info2Expiration	Private	-	-

4.3 Vulnerability Summary

[N1] [Suggestion] Lack of judgment on the balance of withdrawal users

Category: Others

Content

- contracts/lib/Funding.sol

`_requestWithdraw` it is not judged whether the user has a balance, the judgment is placed in `_withdraw`, and users who do not have a balance should be filtered out when the withdrawal is initiated.

```
    function _requestWithdraw(
        Types.State storage state,
        uint256 primaryAmount,
        uint256 secondaryAmount
    ) public {
        if (primaryAmount > 0) {
            state.pendingPrimaryWithdraw[msg.sender] = primaryAmount;
        }
        if (secondaryAmount > 0) {
            state.pendingSecondaryWithdraw[msg.sender] = secondaryAmount;
        }
        state.withdrawExecutionTimestamp[msg.sender] =
            block.timestamp +
            state.withdrawTimeLock;
        emit RequestWithdraw(
            msg.sender,
            primaryAmount,
            secondaryAmount,
            state.withdrawExecutionTimestamp[msg.sender]
        );
    }
```

Solution

The balance withdrawn by the user is compared with the deposit. If the balance is insufficient, the user will not be allowed to apply for withdrawal

Status

Ignored; In line with the project's design expectations.

[N2] [Low] Order validation can be bypassed

Category: Others

Content

- contracts/lib/Trading.sol

The attacker calls the `trade` function in any registered `Perpetual` contract, `order.signer` can be a malicious external address, so that the returned result is always successful.

```
function _validateOrder(
        bytes32 domainSeparator,
        Types.Order memory order,
        bytes memory signature
    ) private returns (bytes32 orderHash) {
        orderHash = EIP712._hashTypedDataV4(
            domainSeparator,
            _structHash(order)
        );
        // contract as trader
        if (Address.isContract(order.signer)) {
            require(
                ISubaccount(order.signer).isValidPerpetualOperator(
                    ECDSA.recover(orderHash, signature)
                ),//SlowMist//order.signer can make the return result true if it is a malicious contract address.
                Errors.INVALID_ORDER_SIGNATURE
            );
        } else {
            require(
                ECDSA.recover(orderHash, signature) == order.signer,
                Errors.INVALID_ORDER_SIGNATURE
            );
        }
        require(
            _info2Expiration(order.info) >= block.timestamp,
            Errors.ORDER_EXPIRED
        );
        require(
            (order.paperAmount < 0 && order.creditAmount > 0) ||
                (order.paperAmount > 0 && order.creditAmount < 0),
            Errors.ORDER_PRICE_NEGATIVE
        );
    }
```

Solution

Validate order.signer

Status

Fixed; record the proxy authority directly in the Dealer contract

[N3] [Low] Risk of excessive authority

Category: Authority Control Vulnerability

Content

- contracts/impl/JOJODealer.sol

`owner` authority is too large The owner can set the `setPerpRiskParams` , `updateFundingRate` in the JOJOOperation module

In particular, `markPriceSource` is the price acquisition address. If the private key of the owner address is lost, the attacker can update the price acquisition address by setting setPerpRiskParams.

After manipulating the price, you can withdraw the contract money,

Solution

It is recommended to transfer the permissions of roles with excessive permissions to governance contracts or timelock contracts. At least multisig should be used.

Status

Ignored; The project party may upgrade the perpetual contract in the future.

[N4] [Suggestion] The perp address is not verified

Category: Others

Content

- contracts/impl/JOJOOperation.sol

Lack of verification of `perp` address.

```
function setPerpRiskParams(address perp, Types.RiskParams calldata param)
    external
    onlyOwner
{
    if (state.perpRiskParams[perp].isRegistered && !param.isRegistered) {
        // remove perp
        for (uint256 i; i < state.registeredPerp.length; i++) {
            if (state.registeredPerp[i] == perp) {
                state.registeredPerp[i] = state.registeredPerp[
                    state.registeredPerp.length - 1
                ];
                state.registeredPerp.pop();
            }
        }
    }
    if (!state.perpRiskParams[perp].isRegistered && param.isRegistered) {
        // new perp
        state.registeredPerp.push(perp);
    }
    require(
        param.liquidationThreshold < 10**18 &&
            param.liquidationPriceOff < param.liquidationThreshold &&
            param.insuranceFeeRate < param.liquidationThreshold,
        Errors.INVALID_RISK_PARAM
    );
    state.perpRiskParams[perp] = param;
    emit UpdatePerpRiskParams(perp, param);
}
```

Solution

The interface function `setPerpRiskParams` can be set by a factory class similar to `Perpetual` to ensure that the perp address must be a contract address that meets the official requirements. Otherwise, if a malicious perp address is set, it will cause serious consequences.

Status

Ignored; The project party may upgrade the perpetual contract in the future.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002204260004	SlowMist Security Team	2022.04.18 - 2022.04.26	Passed

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 low risk, 3 suggestion vulnerabilities. 1 low risk vulnerabilities were ignored; 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