Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

On 2022.12.23, the SlowMist security team received the team's security audit application for JOJOexchange, 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.
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
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

Serial Number	Audit Class	Audit Subclass
		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:

https://github.com/JOJOexchange/smart-contract-EVM/tree/slowmist-audit

commit: 6b0f68a571f8fb547d36bc7483db48e5190a91c6

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Risk of excessive authority	Authority Control Vulnerability Audit	Low	Acknowledged
N2	Token compatibility issue	Others	Suggestion	Acknowledged
N3	Dev address setting enhancement suggestions	Others	Suggestion	Acknowledged

4 Code Overview

4.1 Contracts Description

Codebase:

Audit Version:

https://github.com/JOJOexchange/smart-contract-EVM/tree/slowmist-audit

commit: 6b0f68a571f8fb547d36bc7483db48e5190a91c6

The main network address of the contract is as follows:

Contract Name and Address (Network: Arbitrum)	
Contract Name	Contract Address
Dealer	0x46fdc15E3e2d79b508e6D70E68E9A7127071EE04
Liquidation	0x03Df989dE14F38463bdb70eeef316f797060A351
Funding	0x855E20ce9823d1Add137774fF58b3dbf4d820171
Operation	0xa04A9fedb34BEa5c38042C9848A0fc56729606F9
Perpetual	0xAAE987a9A39d6fCf86BC64D4Dda582dfD4754528
ChainlinkExpandAdaptor	0xc830c8637cA4ED48FcB98B1250aA9817EE4f2c0d
Perpetual	0x9313e231d8B571561F2bB7b7A3C65426d426c906

Contract Name and Address (Network: Arbitrum)	
ChainlinkExpandAdaptor	0x3D031f42B90935fE7BADa8F7724fcf467ab70500
SubaccountFactory	0x54e52aB696767D343D40e8ce0bB78Aa385d6EdFf
FundingRateUpdateLimiter	0x009dbAA6422043b0E7ca8Ea31770e13e834Aa551

4.2 Visibility Description

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

ChainlinkExpandAdaptor			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getMarkPrice	External	-	-

ConstOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getMarkPrice	External	-	-

EmergencyOracle			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	Ownable
getMarkPrice	External	-	-
setMarkPrice	External	Can Modify State	onlyOwner

EmergencyOracleFactory			
Function Name	Visibility	Mutability	Modifiers

EmergencyOracleFactory			
newEmergencyOracle	External	Can Modify State	-

FundingRateUpdateLimiter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
updateFundingRate	External	Can Modify State	onlyOwner
getMaxChange	Public	-	-

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

JOJOView			
Function Name	Visibility	Mutability	Modifiers
getRiskParams	External	-	-
getAllRegisteredPerps	External	-	-
getMarkPrice	External	-	-
getPositions	External	-	-
getCreditOf	External	-	-
isOrderSenderValid	External	-	-
isOperatorValid	External	-	-
isSafe	External	-	-
isAllSafe	External	-	-

JOJOView			
getFundingRate	External	-	-
getTraderRisk	External	-	-
getLiquidationPrice	External	-	-
getLiquidationCost	External	-	-
getOrderFilledAmount	External	-	-
getSetOperatorCallData	External	-	-
getRequestWithdrawCallData	External	-	-
getExecuteWithdrawCallData	External	-	-

JOJOStorage			
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
setOperator	External	Can Modify State	-
handleBadDebt	External	Can Modify State	-
requestLiquidation	External	Can Modify State	onlyRegisteredPerp
openPosition	External	Can Modify State	onlyRegisteredPerp
realizePnl	External	Can Modify State	onlyRegisteredPerp
approveTrade	External	Can Modify State	onlyRegisteredPerp

JOJOExternal			

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
init	External	Can Modify State	-

Subaccount			
execute	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] [Low] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.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:

https://github.com/JOJOexchange/smart-contract-EVM/blob/6b0f68a571f8fb547d36bc7483db48e5190a91c6/contracts/fundingRateKeeper/FundingRateUpdateLimiter.sol#L38-54

```
function updateFundingRate(
    address[] calldata perpList,
    int256[] calldata rateList
) external onlyOwner {
    for (uint256 i = 0; i < perpList.length; i++) {
        address perp = perpList[i];
        int256 oldRate = IPerpetual(perp).getFundingRate();
        uint256 maxChange = getMaxChange(perp);
        require(
            (rateList[i] - oldRate).abs() <= maxChange,
```

```
                "FUNDING_RATE_CHANGE_TOO_MUCH"
            );
            fundingRateUpdateTimestamp[perp] = block.timestamp;
        }

        IDealer(dealer).updateFundingRate(perpList, rateList);
    }
```

2.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 and do not check whether the address is 0 address. This will have a direct impact on perpetual contracts and users' funds.

Code location:

https://github.com/JOJOexchange/smart-contract-EVM/blob/6b0f68a571f8fb547d36bc7483db48e5190a91c6/contracts/impl/JOJOOperation.sol#L33-66

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

    function setFundingRateKeeper(address newKeeper) external onlyOwner {
        Operation.setFundingRateKeeper(state, newKeeper);
    }

    function setWithdrawTimeLock(uint256 newWithdrawTimeLock)
        external
        onlyOwner
    {
        Operation.setWithdrawTimeLock(state, newWithdrawTimeLock);
    }

    /// @notice Secondary asset can only be set once.
    /// Secondary asset must have the same decimal with primary asset.
    function setSecondaryAsset(address _secondaryAsset) external onlyOwner {
        Operation.setSecondaryAsset(state, _secondaryAsset);
    }
```

3.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:

https://github.com/JOJOexchange/smart-contract-EVM/blob/6b0f68a571f8fb547d36bc7483db48e5190a91c6/contracts/impl/Perpetual.sol#L98-113

```
    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

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. And the authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Acknowledged; After communication with the project team, they expressed that they will use a 2 of 3 multi-signer to manage the owner, orderSender they 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.

[N2] [Suggestion] Token compatibility issue

Category: Others

Content

In the `JOJOExternal` contract, users can deposit and withdraw their funds(Primary Asset & Secondary Asset) through the `deposit` and `executeWithdraw` functions. The process of deposit and withdraw call the `safeTransfer` and `safeTransferFrom` functions directly in the `Funding` contract in lib to transfer tokens, and then record the exact amount at the time of transfer. If the `primaryAsset` or `secondaryAsset` ERC20 tokens are deflationary tokens (or other tokens that require a transfer fee) the actual amount of tokens received by the contract or to address will be less than the amount recorded by the amount parameter.

Code location:

https://github.com/JOJOexchange/smart-contract-EVM/blob/6b0f68a571f8fb547d36bc7483db48e5190a91c6/contracts/lib/Funding.sol#L65, L73, L141, L149

```
    function deposit(
        Types.State storage state,
        uint256 primaryAmount,
        uint256 secondaryAmount,
        address to
    ) external {
        if (primaryAmount > 0) {
            IERC20(state.primaryAsset).safeTransferFrom(
                msg.sender,
                address(this),
                primaryAmount
            );
            state.primaryCredit[to] += SafeCast.toInt256(primaryAmount);
        }
        if (secondaryAmount > 0) {
            IERC20(state.secondaryAsset).safeTransferFrom(
                msg.sender,
                address(this),
                secondaryAmount
            );
            state.secondaryCredit[to] += secondaryAmount;
        }
        emit Deposit(to, msg.sender, primaryAmount, secondaryAmount);
    }

    function _withdraw(
        Types.State storage state,
        address payer,
        address to,
        uint256 primaryAmount,
        uint256 secondaryAmount,
        bool isInternal
```

```
    ) private {
        if (primaryAmount > 0) {
            state.primaryCredit[payer] -= SafeCast.toInt256(primaryAmount);
            if (isInternal) {
                state.primaryCredit[to] += SafeCast.toInt256(primaryAmount);
            } else {
                IERC20(state.primaryAsset).safeTransfer(to, primaryAmount);
            }
        }
        if (secondaryAmount > 0) {
            state.secondaryCredit[payer] -= secondaryAmount;
            if (isInternal) {
                state.secondaryCredit[to] += secondaryAmount;
            } else {
                IERC20(state.secondaryAsset).safeTransfer(to, secondaryAmount);
            }
        }
       ...
    }
```

Solution

It is recommended to record the difference between the contract balance before and after the token transfer as the actual transfer amount. Or do not permit such ERC20 tokens in the token list.

Status

Acknowledged; After communication with the project team, they expressed that The secondaryAsset is USDJ, it is an ERC20 standard token. So it won't appear the potential transfer fees. The primaryAsset is USDT, it is also a standard token.

[N3] [Suggestion] Dev address setting enhancement suggestions

Category: Others

Content

In the `JOJOOperation` contract, the owner role can set the insurance in the `setInsurance` function to receive the `insuranceFee`. If the insurance address is an EOA address, in a scenario where the private key is leaked, the team's revenue will be stolen.

Code location:

https://github.com/JOJOexchange/smart-contract-EVM/blob/6b0f68a571f8fb547d36bc7483db48e5190a91c6/contracts/impl/JOJOOperation.sol#L44-46

```
function setInsurance(address newInsurance) external onlyOwner {
    Operation.setInsurance(state, newInsurance);
}
```

Solution

It is recommended to set the insurance address as a multi-signature contract to avoid the leakage of private keys and the theft of team rewards.

Status

Acknowledged; After communication with the project team, they expressed that the current insurance account is an EOA account managed by the JOJO team and we will keep it safe. In the future, the insurance account will be upgraded to a smart contract.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002301130001	SlowMist Security Team	2022.12.23 - 2023.01.13	Low Risk

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, 2 suggestion vulnerabilities. All the findings were fixed.

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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist