Smart Contract
Security Audit Report

Table Of Contents

1 Executive Summary

2 Audit Methodology

3 Project Overview

3.1 Project Introduction

3.2 Vulnerability Information

4 Code Overview

4.1 Contracts Description

4.2 Visibility Description

4.3 Vulnerability Summary

5 Audit Result

6 Statement

1 Executive Summary

On 2022.04.28, the SlowMist security team received the Bhavish team's security audit application for Bhavish finance, 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:

https://github.com/Bhavish-finance/prediction-contract

commit: 833c8305b21961eee2cb2effa7db351178bf29ea

The following contract files are not within the scope of this audit:

- prediction-contract/contracts/Integrations/MinimalForwarder/MinimalForwarder.sol
- prediction-contract/contracts/Integrations/Swap/BhavishSwap.sol
- prediction-contract/contracts/Libs/DateTimeLibrary.sol
- prediction-contract/contracts/Impl/BhavishSDK.sol

3.2 Vulnerability Information

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

NO	Title	Category	Level	Status
N1	Business logic is not clear	Others	Low	Confirmed
N2	NotContract can be bypassed	Others	Low	Fixed
N3	Enhanced event logging	Others	Suggestion	Confirmed
N4	admin address management security reminder	Others	Suggestion	Fixed
N5	Excessive authority issues	Authority Control Vulnerability	Low	Confirmed
N6	roundTime cannot be too short	Design Logic Audit	Low	Confirmed

NO	Title	Category	Level	Status
N7	zero round security reminder	Design Logic Audit	Suggestion	Confirmed

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:

ChainlinkPredicitionOps			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
checkUpkeep	External	-	-
performUpkeep	External	Can Modify State	-

GelatoPredictionOps			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
checker	External	-	-

PredictionOpsManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setPredicitionMarket	Public	Can Modify State	onlyOwner
execute	Public	Can Modify State	-
canPerformTask	External	-	-

BhavishAdministrator			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
<Receive Ether>	External	Payable	-
setAdmin	External	Can Modify State	onlyAdmin
claimTreasury	External	Can Modify State	nonReentrant onlyAdmin notContract

BhavishPrediction			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setOperator	External	Can Modify State	onlyAdmin
setBhavishAdministrator	External	Can Modify State	onlyAdmin
removeOperator	External	Can Modify State	onlyAdmin
setBhavishSwap	External	Can Modify State	onlyAdmin

BhavishPrediction			
<Receive Ether>	External	Payable	-
pause	External	Can Modify State	whenNotPaused onlyAdmin
unPause	External	Can Modify State	whenPaused onlyAdmin
setMinPredictAmount	External	Can Modify State	whenPaused onlyAdmin
setTreasuryFee	External	Can Modify State	whenPaused onlyAdmin
setRoundTime	External	Can Modify State	whenPaused onlyOperator
_createRound	Private	Can Modify State	-
_startRound	Private	Can Modify State	-
_cancelRound	Private	Can Modify State	-
_endRound	Private	Can Modify State	-
_calculateRewards	Private	Can Modify State	-
_safeTransfer	Private	Can Modify State	-
_refundable	Private	-	-
transferToAdmin	External	Payable	nonReentrant onlyAdmin
predictUp	External	Payable	whenNotPaused nonReentrant
predictDown	External	Payable	whenNotPaused nonReentrant
createPredictionMarket	External	Can Modify State	whenNotPaused onlyOperator notContract

BhavishPrediction			
startPredictionMarket	External	Can Modify State	whenNotPaused onlyOperator notContract
_claimable	Public	-	-
claim	External	Can Modify State	nonReentrant
getRewards	Public	-	-
_calcRewardsForUser	Private	-	-
refundUsers	Public	Can Modify State	nonReentrant
executeRound	External	Can Modify State	whenNotPaused
getCurrentRoundDetails	External	-	-

BhavishPredictionStorage			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setManager	External	Can Modify State	-
createPredictionRound	External	Can Modify State	onlyManager
updatePredictionRound	External	Can Modify State	onlyManager
getUsersInRounds	External	-	-
getPredictionRound	External	-	-
getArrayRounds	External	-	-
setRoundState	External	Can Modify State	onlyManager

BhavishPredictionStorage			
cancelRound	External	Can Modify State	onlyManager
setRewardAmountForRound	External	Can Modify State	onlyManager
setAmount	External	Can Modify State	onlyManager
createBet	External	Can Modify State	onlyManager
getBetInfo	External	-	-
setBetAmountDispersed	External	Can Modify State	onlyManager
setBetInfo	External	Can Modify State	onlyManager
setLedgerInfo	External	Can Modify State	onlyManager
getUserRoundHistory	External	-	-

PriceManager			
Function Name	Visibility	Mutability	Modifiers
setManager	Public	Can Modify State	-
removeManager	Public	Can Modify State	-
<Constructor>	Public	Can Modify State	-
setRoundIdChecks	External	Can Modify State	onlyAdmin
setRoundIdTimestampDifference	External	Can Modify State	onlyAdmin
getPrice	Public	-	-
setPairContract	Public	Can Modify State	onlyManager
getLatestRoundId	Internal	-	-

PriceManager			
getPriceByTimestamp	External	-	-
getTimestampByRoundId	External	-	-

BaseRelayRecipient			
Function Name	Visibility	Mutability	Modifiers
isTrustedForwarder	Public	-	-
msgSender	Internal	-	-

4.3 Vulnerability Summary

[N1] [Low] Business logic is not clear

Category: Others

Content

The execute() function does not accept parameters, but abi.encodeWithSignature("execute()", predictionOpsManager) inputs predictionOpsManager, the business logic here is not clear.

- contracts/Automation/GelatoPredictionOps.sol#L18

```
    function checker() external view override returns (bool canExec, bytes memory execPayload) {
        // solhint-disable not-rely-on-time
        canExec = predictionOpsManager.canPerformTask();

        execPayload = abi.encodeWithSignature("execute()", predictionOpsManager);
    }
```

When executing the for loop, it should be ensured that the canPerform returned by each round is true, but the for loop will be terminated once one of them returns true. So it should be if (!canPerform) break?

- contracts/Automation/PredictionOpsManager.sol#L40-L50

```
function canPerformTask() external view returns (bool canPerform) {
        for (uint256 i = 0; i < predictionMarkets.length; i++) {
            IBhavishPrediction.Round memory round =
predictionMarkets[i].getCurrentRoundDetails();

            canPerform =
                block.timestamp > round.roundEndTimestamp &&
                round.roundState != IBhavishPrediction.RoundState.ENDED;

            if (canPerform) break;
        }
    }
```

Solution

It is recommended to communicate with the developers the business logic implemented by the code here.

Status

Confirmed; The project team response: the canPerform flag is used to understand if atleast one of the prediction market is ready for creating the rounds.

If the condition is true, then execute() method of the same contract will be invoked wherein the executeRound() of the prediction market is invoked based on the same conditions.

The SlowMist security team response: After communication with project team, there is no security risk in this issue.

[N2] [Low] NotContract can be bypassed

Category: Others

Content

NotContract uses Address.sol:isContract for judgment, which can be bypassed. When the contracts in construction the code.length is 0.

- contracts/Impl/BhavishAdministrator.sol#L29-L32

```
    modifier notContract(address _address) {
        require(!address(_address).isContract(), "Contract not allowed");
        _;
    }
```

- node_modules/@openzeppelin/contracts/utils/Address.sol#L39-L42

```
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize/address.code.length, which returns 0
        // for contracts in construction, since the code is only stored at the end
        // of the constructor execution.

        return account.code.length > 0;
    }
```

Solution

It is recommended to use tx.origin == msg.sender to determine the EOA address.

Status

Fixed; The project team fixed this issue by removing notContract modifier.

[N3] [Suggestion] Enhanced event logging

Category: Others

Content

When recording events, only new values are recorded, which is not conducive to the review of community users.

- contracts/Impl/BhavishPrediction.sol#L195-L212

```
    function setMinPredictAmount(uint256 _minPredictAmount) external whenPaused
onlyAdmin(msg.sender) {
        require(_minPredictAmount > 0, "Must be superior to 0");
        minPredictAmount = _minPredictAmount;

        emit NewMinPredictAmount(_minPredictAmount);
    }
```

```
    function setTreasuryFee(uint256 _treasuryFee) external whenPaused
onlyAdmin(msg.sender) {
        require(_treasuryFee > 0 && _treasuryFee < MAX_TREASURY_FEE, "Treasury fee is
too high");
        treasuryFee = _treasuryFee;

        emit NewTreasuryFee(_treasuryFee);
    }
```

- contracts/Impl/BhavishAdministrator.sol#L44-L49

```
    function setAdmin(address _admin) external onlyAdmin(msg.sender) {
        require(_admin != address(0), "Cannot be zero address");
        grantRole(DEFAULT_ADMIN_ROLE, _admin);

        emit NewAdmin(_admin);
    }
```

- contracts/Impl/BhavishPrediction.sol#L120-L126

```
    function setOperator(address _operator) external onlyAdmin(msg.sender) {
        require(!address(_operator).isContract(), "Operator cannot be a contract");
        require(_operator != address(0), "Cannot be zero address");
        grantRole(OPERATOR_ROLE, _operator);

        emit NewOperator(_operator);
    }
```

Solution

It is recommended to record old and new values to facilitate review by community users.

Status

Confirmed; The project team response: We index old data in subgraph, so no need to emit events for old values.

[N4] [Suggestion] admin address management security reminder

Category: Others

Content

The admin will manage treasuryFee. There is a risk of centralized authority management.

- contracts/Impl/BhavishAdministrator.sol#L58-L68

```
    function claimTreasury(uint256 _amount)
        external
        override
        nonReentrant
        onlyAdmin(msg.sender)
        notContract(msg.sender)
    {
        require(_amount <= address(this).balance, "Cannot claim > contract balance");
        (bool success, ) = admin.call{ value: _amount }("");
        require(success, "TransferHelper: TRANSFER_FAILED");

        emit TreasuryClaim(admin, _amount);
    }
```

After the admin is assigned, it cannot be modified. If the private key of the admin is lost or leaked, it will lead to loss of fee income.

- contracts/Impl/BhavishAdministrator.sol#L44-L49

```
    function setAdmin(address _admin) external onlyAdmin(msg.sender) {
        require(_admin != address(0), "Cannot be zero address");
        grantRole(DEFAULT_ADMIN_ROLE, _admin);

        emit NewAdmin(_admin);
    }
```

- contracts/Impl/BhavishAdministrator.sol#L20

```
constructor() {
        admin = payable(msg.sender);
```

```
        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
    }
```

Solution

It is recommended to set admin to multi-sign contracts to avoid the loss of funds caused by the loss of private keys.

Status

Fixed; The project team response: Initially most of the admin operations will be managed by Bhavish and then we slowly move to governance. So, the risk is very less.

[N5] [Low] Excessive authority issues

Category: Authority Control Vulnerability

Content

Admin can setPairContract, if the wrong contract address is set, it will affect the price obtained. Wrong price affects user profitability.

- contracts/Impl/PriceManager.sol#L76-L85

```
    function setPairContract(
        bytes32 _underlying,
        bytes32 _strike,
        address _aggregator
    ) public override onlyManager(msg.sender) {
        require(_aggregator.isContract(), "Chainlink: Invalid aggregator");
        addressMap[_underlying][_strike] = _aggregator;

        emit AddAssetPairAggregator(_underlying, _strike, address(this),
_aggregator);
    }
```

Missing event records, and Admin can change the parameters of the contract.

- contracts/Impl/PriceManager.sol#L46-L53

```
    function setRoundIdChecks(uint8 _limitChecks) external onlyAdmin(msg.sender) {
        require(_limitChecks >= 0 && _limitChecks <= 10, "invalid limit range");
        roundIdChecks = _limitChecks;
    }

    function setRoundIdTimestampDifference(uint256 _difference) external
onlyAdmin(msg.sender) {
        roundIdTimestampDifference = _difference;
    }
```

Solution

It is recommended to set Admin to a timelock contract or a multi-signature contract. And add event logging for setRoundIdTimestampDifference and setRoundIdChecks.

Status

Confirmed; The project team response: We have taken out the redundant methods. Also, the setter is used for 'numItr' state variable which is being handled by admin initially and slowly we will be moving to governance.

[N6] [Low] roundTime cannot be too short

Category: Design Logic Audit

Content

roundTime cannot be too short, and cannot be less than the price update time of chainlink.

- contracts/Impl/BhavishPrediction.sol#L219-L222

```
    function setRoundTime(uint256 _roundTime) external whenPaused
onlyOperator(msg.sender) {
        require(0 <= _roundTime && _roundTime <= 3600, "Round Time should be between
1 sec to 3600 sec");
        roundTime = _roundTime;
    }
```

Solution

It is recommended to refer to the price update time of chainlink for configuration.

Reference:

https://docs.chain.link/docs/ethereum-addresses/

https://docs.chain.link/docs/historical-price-data/#:~:text=getRoundData

Status

Confirmed; The project team response: Price interval update is based on deviation and heartbeat, Since we can't get those details on chain.

[N7] [Suggestion] zero round security reminder

Category: Design Logic Audit

Content

The startPredictionMarket function should be called in time during the zero round, otherwise the round time will be shortened and the price may be predicted.

- contracts/Impl/BhavishPrediction.sol#L459-L470

```
        function startPredictionMarket() external override whenNotPaused
onlyOperator(msg.sender) {
        require(marketStatus.createPredictionMarketOnce, "Can only run after
roundzeroCreateRound is triggered");
        require(!marketStatus.startPredictionMarketOnce, "Can only run
roundzeroStartRound once");

        require(block.timestamp >= roundzeroStartTimestamp, "Round cannot be started
early");

        (uint256 price, ) = bhavishPriceManager.getPrice(
            assetPair.underlying,
            assetPair.strike,
            roundzeroStartTimestamp
        );

        _startRound(currentRoundId, price);
```

```
        marketStatus.startPredictionMarketOnce = true;
    }
```

Solution

It is recommended to call the startPredictionMarket function in time in the zero round.

Status

Confirmed; The project team response: We are calling the round zero at the specified time only and also, there will be no prediction in the initial rounds.

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002205100003	SlowMist Security Team	2022.04.28 - 2022.05.10	Low Risk

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 4 low risk, 3 suggestion vulnerabilities. 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.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist