Smart Contract Security Audit Report

Contents

1. Executive Summary

On Dec. 21, 2020, the SlowMist security team received the dFuture team's security audit application for dFuture , 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

Initial audit version：

dFuture-core:

https://github.com/dFuture-finance/dFuture-core

commit: 3580d39a5db7d28487b8ed406f95e6081ebaa88f

dfuture-oracle-aggeregator

https://github.com/dFuture-finance/dFuture-core

commit: 5c2ad0483f457b43befaeacbcd236c483373d67e

Final audit version：

dFuture-core:

https://github.com/dFuture-finance/dFuture-core

commit: 1ce02f9375812da78f72bf8ce9ee4783660b7a45

dfuture-oracle-aggeregator

https://github.com/dFuture-finance/dFuture-core

commit: 6a4a69a3a38868e0e9abb314f46fd689d56aece9

Audit version file information

3.2 Project Structure

dFuture-core:

```
contracts
├── FPView.sol
├── FuturePerpetual.sol
├── LpTokenWrapper.sol
├── Migrations.sol
├── MockPool.sol
├── Timelock.sol
├── impl
│   ├── FPChecker.sol
│   ├── FPDistribute.sol
│   ├── FPInterests.sol
│   ├── FPLedger.sol
│   ├── FPOracleWrapper.sol
│   ├── FPPoolWrapper.sol
│   ├── FPPosition.sol
│   ├── FPSharePool.sol
│   ├── FPStorage.sol
│   ├── LPWrapperStorage.sol
│   └── Log.sol
├── intf
│   ├── AggregatorV3Interface.sol
│   ├── ILpToken.sol
│   ├── ILpTokenWrapper.sol
│   ├── IOracle.sol
```

```
|   ├── IOracleAggregator.sol
|   ├── IPool.sol
|   ├── IUniswapV2Pair.sol
|   └── IUniswapV2Router02.sol
├── lib
|   ├── FPTypes.sol
|   ├── MixedMath.sol
|   └── PositionUtil.sol
├── oracle
|   ├── AggregatorV3Interface.sol
|   ├── MockChainLink.sol
|   └── MockOracleAggeregator.sol
├── pool
|   ├── Pool.sol
|   └── PoolBase.sol
└── tokens
    ├── DToken.sol
    ├── LpToken.sol
    └── MockUsdtToken.sol
```

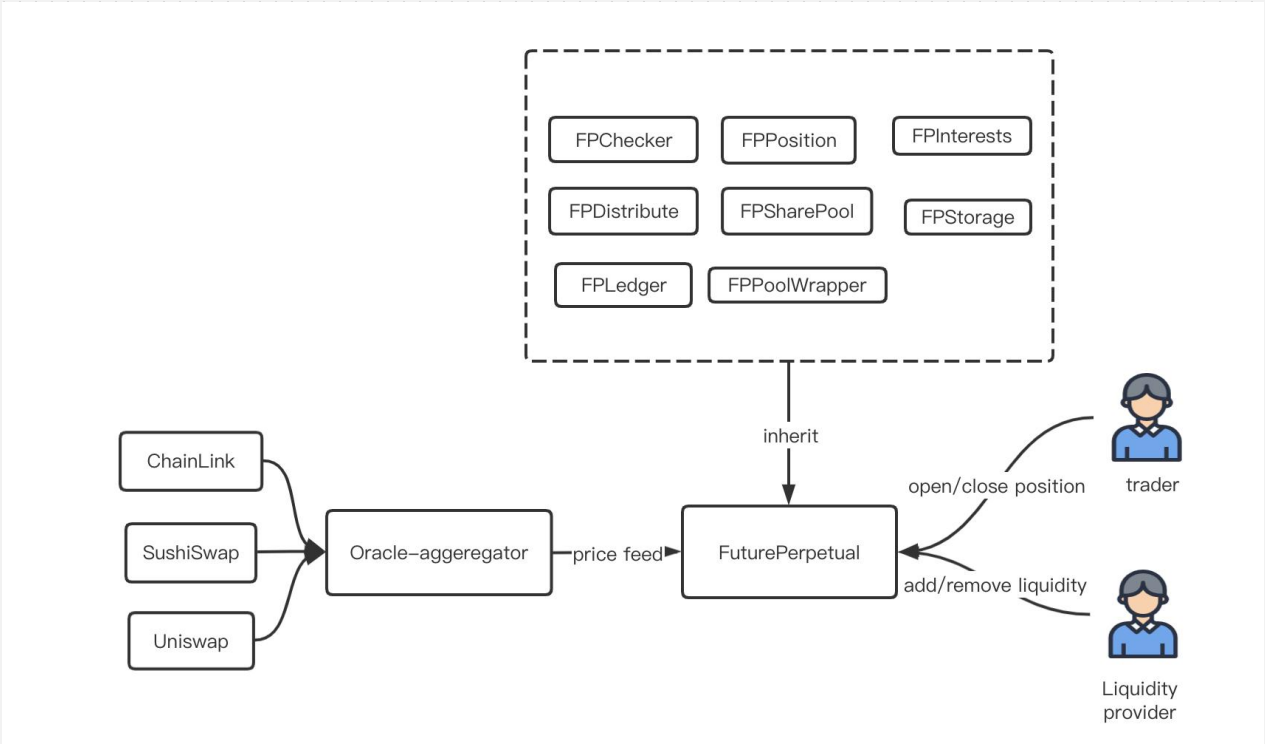
dfuture-oracle-aggeregator

```
contracts
├── Migrations.sol
├── intf
|   ├── AggregatorV3Interface.sol
|   ├── IOracle.sol
|   ├── IOracleAggregator.sol
|   ├── IUniswapV2Pair.sol
|   └── IUniswapV2Router02.sol
├── lib
|   ├── Babylonian.sol
|   ├── BitMath.sol
|   ├── FPTypes.sol
|   ├── FixedPoint.sol
|   ├── FullMath.sol
|   ├── MixedMath.sol
|   ├── UQ112x112.sol
|   └── UniswapV2OracleLibrary.sol
└── oracles
    ├── ChainLinkOracle.sol
    ├── MockChainLink.sol
```

```
├── MockUniswap.sol
├── OracleAggregator.sol
├── OracleBase.sol
├── SushiswapInstant.sol
├── UniswapInstant.sol
└── UniswapOracle.sol
```

3.3 Contract Structure

The dFuture contract mainly contains two parts, namely the FuturePerpetual contract, the main logic part and the Oracle-aggeregator contract, which are responsible for obtaining the price of the corresponding trading pair from ChainLink, SushiSwap and Uniswap. Traders can open and close positions through the FutruePerpetual contract and earn profits. Liquidity providers can earn transaction fees and mining revenue by providing liquidity. The overall system architecture diagram is as follows:



4. Code Overview

4.1 Contract Visibility Analysis

LpTokenWrapper			
Function name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
initialize	External	can modify state	onlyOnce
transferOwnership	External	can modify state	onlyOwner
setMaster	External	can modify state	onlyOwner
setAddress	External	can modify state	onlyOwner
setUsdtAddress	External	can modify state	onlyMaster
setDftUsdtRatio	External	can modify state	onlyOwner
stakeDToken	External	can modify state	-
unstakeDToken	External	can modify state	-
getStakedLpTokenInfo	External	-	-

getLpTokenInfoWithAccelerate	External	-	-
getStakedDTokenInfo	External	-	-
getPoolUserInfo	Private	-	
getAccelerateInfo	External	-	-
pendingToken	External	-	-
pendingTokenInternal	Private	-	-
allPendingToken	External	-	-
claimAllPendingToken	External	can modify state	-
claimPendingToken	External	can modify state	-
isSpecialPool	Private	-	-
pendingTokenForSpecialPool	External	-	-
claimFomoToken	External	can modify state	onlyMaster
claimFomoToSelf	Private	can modify state	-
claimSpecialPoolBatch	External	can modify state	-
claimSpecialPool	External	can modify state	-

specialIndex2AddrIndex	Private	-	-
poolSize	External	-	-
setIssueCheckPoint	External	can modify state	onlyOwner
setParent	External	can modify state	-
setRoot	External	can modify state	onlyOwner
getParent	External	-	-
onRelationCheck	External	can modify state	onlyMaster
setRelationForce	External	can modify state	onlyOwner
adjustPoolParam	External	can modify state	onlyOwner
settleAllIssuePool	Private	can modify state	-
setRelation	Private	can modify state	-
addAccelerate	External	can modify state	-
removeAccelerate	External	can modify state	-
updateAcceleratePoolUserInfo	Private	can modify state	-
updateAcceleratePool	Private	can modify state	-

isFirstMint	External	-	-
distributeUsdtByFees	External	can modify state	onlyMaster
mint	External	can modify state	onlyMaster
burn	External	can modify state	onlyMaster
getRatioOf	External	-	-
getTotalRatioOf	External	-	-
getLpTokenAmount	External	-	-
stake	Private	can modify state	-
unstake	Private	can modify state	-
updateIssuePoolUserInfo	Private	can modify state	-
updateFeePoolUserInfo	Private	can modify state	-
settleUserReward	Private	can modify state	-
updatePoolUserAmount	Private	can modify state	-
updateUserDebt	Private	can modify state	-
updatePoolInfo	Private	can modify state	-

claimPoolDToken	Private	can modify state	-
poolDTokenPendingInternal	Private	-	-
safeMintDToken	Private	can modify state	-
safeDTokenTransfer	Private	can modify state	-
safeUsdtTransfer	Private	can modify state	-
safeTransfer	Private	can modify state	-
getIssuedDTokenFromLastTo-w	Private	-	-
getWorkIndex	Private	-	-
updateAcceleratePool	Private	can modify state	-
getPoolTokenAmount	Private	-	-
getOwnerTokenAmount	Private	-	-

DToken			
Function name	Visibility	Mutability	Modifiers

constructor	Public	can modify state	ERC20
mint	Public	can modify state	-
pause	Public	can modify state	-
unpause	Public	can modify state	-

LpToken			
Function name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	ERC20
mint	External	can modify state	onlyOwner
burn	External	can modify state	onlyOwner

FuturePerpetual			
Function name	Visibility	Mutability	Modifiers

initialize	External	can modify state	onlyOnce
transferOwnership	External	can modify state	onlyOwner
pause	External	can modify state	onlyOwner
unpause	External	can modify state	onlyOwner
clearing	External	can modify state	onlyOwner
unclearing	External	can modify state	onlyOwner
setExternalAddress	External	can modify state	onlyOwner
setRatio	External	can modify state	onlyOwner
setSupportTradePair	External	can modify state	onlyOwner
setTradePairStatus	External	can modify state	onlyOwner
deposit	External	can modify state	whenNotClearing
withdraw	External	can modify state	whenNotPaused
openPosition	External	can modify state	whenNotPaused
doOpenPosition	Private	can modify state	whenNotClearing
closePosition	External	can modify state	whenNotPaused

closePositionByMaster	External	can modify state	-
doClosePosition	Private	can modify state	-
forceClosePosition	External	can modify state	whenNotPaused
depositToPool	External	can modify state	whenNotClearing
healPool	External	can modify state	-
withdrawFromPool	External	can modify state	whenNotPaused
claimAllUSDT	External	can modify state	whenNotPaused
claimFeeAndInterestInternal	Private	can modify state	-
claimReserved	External	can modify state	whenNotPaused
distributeFeesOfLP	External	can modify state	whenNotPaused
queryIOUFeesAndInterests	External	-	-
queryPosition	External	-	-
queryPositionFeeAndRatio	External	-	-
queryBalanceOfFees	External	-	-
queryHolderInfo	External	-	-

queryFomoInfo	External	-	-
queryInterestRatio	External	-	-
queryPendingInterest	External	-	-
queryPoolDeposit	External	-	-

PositionUtil			
Function name	Visibility	Mutability	Modifiers
mergePosition	Internal	-	-
calProfitAndValue	Internal	-	-
versusDirection	Internal	-	-

MixedMath			
Function name	Visibility	**Mutability**	**Modifiers**

add	Internal	-	-
sub	Internal	-	-
sub	Internal	-	-
add	Internal	-	-
abs	Internal	-	-
opposite	Internal	-	-

Pool			
Function name	Visibility	**Mutability**	**Modifiers**
limitPositionInBlock	Private	can modify state	-
refreshAllSymbolsPrice	Public	can modify state	NO
symbolsAndPrices	Private	can modify state	-
symbolsAndPricesForView	Private	-	-
limitRatioToRange	Private	-	-

initialize	External	can modify state	onlyOnce
transferOwnership	External	can modify state	onlyOwner
setMaster	External	can modify state	onlyOwner
setValueInBlock	External	can modify state	onlyOwner
setRatio	External	can modify state	onlyOwner
setOracleAddress	External	can modify state	onlyOwner
setSupportSymbols	External	can modify state	onlyOwner
marginRatioWithCachedPrice	Internal	can modify state	-
openPosition	External	can modify state	onlyMaster
closePosition	External	can modify state	onlyMaster
addLiquid	External	can modify state	onlyMaster
withdraw	External	can modify state	onlyMaster
subLiquid	External	can modify state	onlyMaster
getPositionOffsetWithSymbol	Public	-	-
getPositionOffset	Public	-	-

getProfitAndValue	Internal	-	-
getPositionFeeRatioWithOffset	Private	-	-
getPositionFeeRatio	Public	-	-
positionOffsetWithCache	Private	can modify state	-
getPositionFeeRatioWithCache	Public	can modify state	-
getInterestRateOf	Public	-	-
getInterestRateOfWithCache	Public	can modify state	-
getMarginRatio	Public	-	-
getInfo	Public	-	-
getPositionOf	Public	-	-
getUsdtAmountByRatio	External	-	-
getRatioByUsdtAmount	External	-	-

FPSharePool

Function name	Visibility	Mutability	Modifiers
depositFees	Internal	can modify state	-
depositInterest	Internal	can modify state	-
distributableLpFeeInterestAmount	Internal	can modify state	-
canClaimIOUFeesAmount	Internal	can modify state	-
canClaimInterestAmount	Internal	can modify state	-
updateFomoInfo	Internal	can modify state	-

FPSharePool			
Function name	Visibility	Mutability	Modifiers
constructor	Public	can modify state	-
receive	External	payable	-
setDelay	Public	can modify state	-
acceptAdmin	Public	can modify state	-

setPendingAdmin	Public	can modify state	-
queueTransaction	Public	can modify state	-
cancelTransaction	Public	can modify state	-
executeTransaction	Public	payable	-
getBlockTimestamp	Internal	-	-

4.2 Code Audit

4.2.1 Low-risk vulnerabilities

4.2.1.1 The settleHolderInterest is not used to update user interest before Withdraw

The code uses payOffInterest to process the user's interest when processing the user's withdraw, but it did not use settleHolderInterest to update the user's interest situation before, resulting in a deviation in the interest payment.

Location: FuturePerpeatual.sol withdraw function

```
function withdraw(bytes32 _symbol, uint256 _usdtAmount)
        external
        whenNotPaused
        //SlowMist// Did not use settleHolderInterest to update the user's interest first
    {
        // Pay off the owned interest if there is any
        FPInterests.payOffInterest(msg.sender, _symbol);

        // Help to trigger the distribution of transaction fee and interest
```

```
        FPDistribute.distributeLpFees();

        subLedgerOfHolder(msg.sender, _symbol, _usdtAmount);

        // After the withdraw, must meet the maintain margin ratio
        if (HolderPositions[msg.sender][_symbol].amount != 0) {
            uint256 currentPrice = readLatestPrice(_symbol);
            uint256 marginRatio = getMarginRatioOf(msg.sender, _symbol, currentPrice);

            require(
                marginRatio > SysRatios[MaintainMarginRatio],
                "Lower than Maintain Margin Ratio"
                );
        }
        // transfer USDT to the trader
        transferUsdtFromPerpetual(msg.sender, _usdtAmount);

        emit Withdraw(msg.sender, _symbol, _usdtAmount);
    }
```

Fix suggestion: It is recommended to use settleholderInterest to update user interest again.

Fix status: After communicating with the project party, it is confirmed that this issue will not be adjusted or repaired in order to save gas costs.

4.2.1.2 The global status is not updated when using the getMarginRatioOf function to calculate the user's position

When using the getMarginRationOf function to calculate the user's position, the global state is not updated with updateGlobalInterestRate first, which may cause calculation errors.

Code location: FPPosition.sol getMarginOf function

```
function getMarginRatioOf(address holder, bytes32 symbol, uint256 currentPrice) internal view returns (uint256) {
        //SlowMist// The global state is not updated, which may cause calculation errors
        FPTypes.Position memory position = HolderPositions[holder][symbol];
        (int256 totalProfit, uint256 totalValue) = PositionUtil.calProfitAndValue(
            position,
            currentPrice);
```

```
        if (totalValue == 0) return MAX_ENOUGH_MARGIN;

        // Margin ratio   = ( profit/loss + balance + owing interest) / position value
        uint128 startIndex = gUserInterestSettledIndex[holder][symbol];
        uint128 updatedIndex = GlobalInterestIndex[symbol].next;
        int256 rate;
        for (uint128 i = startIndex; i < updatedIndex; i++) {
            int96 price = position.direction == ShortDirection ?
                        GlobalInterestRates[symbol].rates[i].shortRatioWithPrice :
                        GlobalInterestRates[symbol].rates[i].longRatioWithPrice;
            rate = rate.add(int256(price));
        }

        int256 interest;
        if (rate > 0) {
            interest = rate.mul(int256(position.amount)).div(int256(PrecisionDecimals6));
        }

        if (IOUInterest[holder] > 0) { interest = interest.add(IOUInterest[holder]); }

        uint256 balance = totalProfit.sub(interest).add(HolderPositionBalances[holder][symbol]);
        return balance.mul(PrecisionDecimals6).div(totalValue);
    }
```

Fix suggestion: update the global status in the getMarginRatioOf function.

Fix status: After communicating with the project party, it is confirmed that getMarginRatioOf is a view method, which is used to call and query margin ratio for external clients, and there is no write operation to the database. At the same time, the system calls settleHolderInterst on all paths that need to call getMarginRatioOf to update the global status (opening, closing, liquidation). This problem will not be repaired.

4.2.1.3 When calculating the user's position, the user's intereset <0 is not considered

When using the getMarginRationOf function to calculate the user's position, the case of rate <0 is not

processed.

Code location: FPPosition.sol getMarginOf function

```
function getMarginRatioOf(address holder, bytes32 symbol, uint256 currentPrice) internal view returns (uint256) {

        FPTypes.Position memory position = HolderPositions[holder][symbol];
        (int256 totalProfit, uint256 totalValue) = PositionUtil.calProfitAndValue(
            position,
            currentPrice);

        if (totalValue == 0) return MAX_ENOUGH_MARGIN;

        // Margin ratio  = ( profit/loss + balance + owing interest) / position value
        uint128 startIndex = gUserInterestSettledIndex[holder][symbol];
        uint128 updatedIndex = GlobalInterestIndex[symbol].next;
        int256 rate;
        for (uint128 i = startIndex; i < updatedIndex; i++) {
            int96 price = position.direction == ShortDirection ?
                        GlobalInterestRates[symbol].rates[i].shortRatioWithPrice :
                        GlobalInterestRates[symbol].rates[i].longRatioWithPrice;
            rate = rate.add(int256(price));
        }

        int256 interest;
        if (rate > 0) {
            interest = rate.mul(int256(position.amount)).div(int256(PrecisionDecimals6));
        } //SlowMist// Does not consider the case where rate is less than 0


        if (IOUInterest[holder] > 0) { interest = interest.add(IOUInterest[holder]); }

        uint256 balance = totalProfit.sub(interest).add(HolderPositionBalances[holder][symbol]);
        return balance.mul(PrecisionDecimals6).div(totalValue);
}
```

Fix suggestion: Decide whether to handle rate <0 according to the business scenario.

Fix status: After communicating with the project party, it is confirmed that the business will not consider such issues for the time being, and the issue will be adjusted through the business situation

in the future.

4.2.2.1 did not consider the issue of system compensation, and did not limit the maximum benefits of users

The system code does not consider whether the system can pay for this when processing the user's position closing. When this happens, it will cause an unknown error.

Code location: Pool.sol subLiquid function

```
function subLiquid(uint256 usdtAmount) external onlyMaster
    {
        PoolUsdtBalances = PoolUsdtBalances.sub(usdtAmount);
    }
```

Fix suggestion: Check the profit amount and set the maximum profit threshold to reduce the risk that the system cannot pay.

Fix status: After communicating with the project party, the project party confirmed that SafeMath will generate a transaction rollback when the system fails to pay, in the form that the user cannot close the position. This issue will be designed more systematically based on business considerations. Prevent the system from failing to pay.

4.2.2 Enhancement Suggestions

5. Audit Result

5.1 Conclusion

Audit Result : Some Risks

Audit Number : 0X002101050002

Audit Date : Jan 05, 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 four security issues found during the audit. There are three low-risk vulnerabilities and one enhancement suggestion. We also provide five enhancement suggestions. 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