SLOWMIST

Smart Contract Security Audit Report

Contents

1. Executive Summary

On Mar. 10, 2021, the SlowMist security team received the Shield team's security audit application for Shield, 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

The risk-free perpetual contract is Shield’s answer to the existing limitations within the decentralized derivative ecosystem. It uses a combination of a dual liquidity pool model, SLD (the native token of the platform), a decentralized brokerage system, and external liquidators to counteract the existing limitations.

Audit files:

https://github.com/Jackluren/DDS-Contract-Test

commit: e3b7ebdd194b544f14cc59ce6694521a1c550676

3.2 Project Structure

```
├── DDSDAIContracts.sol
├── DDSDAIPools1.sol
├── DDSDAIPools2.sol
├── DDSFormular.sol
├── DDSInterfaces.sol
├── DDSUSDCContracts.sol
├── DDSUSDCPools1.sol
├── DDSUSDCPools2.sol
├── DDSUSDTContracts.sol
├── DDSUSDTPools1.sol
├── DDSUSDTPools2.sol
└── back
    ├── DDSContracts.sol
    ├── DDSPools1.sol
    └── DDSPools2.sol
```

4. Code Overview

4.1 Main Contract address

Not yet deployed on the mainnet.

4.2 Contracts Description

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

PublicPool			
Function Name	Visibility	Mutability	Modifiers
provide	Public	Can modify state	-
withdraw	Public	Can modify state	-
lock	Public	Can modify state	onlyKeeeper
moveLp1Fund	Public	Can modify state	onlyLP2Keeeper
close	Public	Can modify state	onlyKeeeper
riskClose	Public	Can modify state	onlyKeeeper
setLockupPeriod	Public	Can modify state	onlyOwner
totalBalance	Public	-	-

getTotalSupply	Public	-	-
getLockedAmount	Public	-	-
getMintReDaiAmount	Public	-	-
getTokenAmountByreToken	Public	-	-
getReTokenAmountByToken	Public	-	-
getMatchID	Public	-	-
geMarginAmount	Public	-	-
getlockedLiquidityLen	Public	-	-
getRiskFundAmount	Public	-	-
getLPAmountInfo	Public	-	-
getUserReToeknInfo	Public	-	-
_safeTransferFrom	Internal	Can modify state	-
_safeTransfer	Internal	Can modify state	-
isEqual	Public	-	-
setFormular	Public	Can modify state	onlyOwner
setPoolTokenAddr	Public	Can modify state	onlyOwner
setRiskFundAddr	Public	Can modify state	onlyOwner
setKeeper	Public	Can modify state	onlyOwner
setLP2Keeper	Public	Can modify state	onlyOwner

PrivatePool			
Function Name	Visibility	Mutability	Modifiers
provide	Public	Can modify state	-
withdraw	Public	Can modify state	-
lock	Public	Can modify state	onlyKeeeper
lockLP2	Internal	Can modify state	-
close	Public	Can modify state	onlyKeeeper
riskClose	Public	Can modify state	onlyKeeeper
addMarginAmount	Public	Can modify state	-
setLockupPeriod	Public	Can modify state	onlyOwner
setIsRejectOrder	Public	Can modify state	-
totalBalance	Public	-	-
getMatchLp2Object	Public	-	-
getRiskFundAmount	Public	-	-
getMatchID	Public	-	-
getlockedLiquidityLen	Public	-	-

geMarginAmount	Public	-	-
getLPAmountInfo	Public	-	-
getLPLocked	Public	-	-
_safeTransferFrom	Internal	Can modify state	-
_safeTransfer	Internal	Can modify state	-
setPoolTokenAddr	Public	Can modify state	onlyOwner
setRiskFundAddr	Public	Can modify state	onlyOwner
setPublicPoolAddr	Public	Can modify state	onlyOwner
setKeeper	Public	Can modify state	onlyOwne

Formular			
Function Name	Visibility	Mutability	Modifiers
sqrt	Internal	-	-
updateImpliedVolrate	External	Can modify state	-
updatePriceByOwner	External	Can modify state	-
updatePrice	Internal	Can modify state	-
calculateimpliedVolrate	Internal	Can modify state	-
getMargin	Public	-	-
getPositionFee	Public	-	-
getOpenFee	Public	-	-
getMaxOpenAmount	External	-	-
isEqual	Internal	-	-

4.3 Code Audit

4.3.1 Critical vulnerabilities

4.3.1.1 Risk of Oracle Manipulation

In DDSContracts, the price is obtained from Uniswap's getAmountsIn through the getPriceByWBTCDAI function, but this interface obtains the real-time price of the WBTC/DAI pool, and there is a risk of malicious manipulation.

Fix suggestion: It is recommended to use Uniswap's delayed price feed oracle for acquisition.

Code location: DDSContracts.sol

```
function getPriceByWBTCDAI() public view returns(uint256){
    address[] memory WbtcToDaiSwapPath;
    WbtcToDaiSwapPath = new address[](2);
    WbtcToDaiSwapPath[0] = address(0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599);
    WbtcToDaiSwapPath[1] = address(0x113587939c8967e61Aa2360613951B23AB2Af49a);
    uint256 amount = 1e8*1e18;
    return uniswapRouter.getAmountsIn(amount, WbtcToDaiSwapPath)[0];
}
```

Fix status: Fixed in this commit: 6aecab747d169512d7150570f61ddfb0f1ee77c9

4.3.1.2 Price acquisition issue when opening and closing positions

In the DDSContracts contract, the price used when opening and closing a position is passed in from the outside, which will cause the user to pass in any price when opening and closing a position. After communicating with the project party, this is the test code, and the oracle will be used to feed the price during the formal deployment.

Fix suggestion: It is recommended to use Uniswap's delayed price feed oracle for acquisition.

Code location: DDS*Contracts.sol

```
function riskControl(uint256[] calldata orderIDs,uint256 currentPrice) public

function closeContract(uint256 orderID,uint256 currentPrice) public

function migrationContract(uint256 orderID,uint256 currentPrice) public
```

Fix status: Fixed in this commit: 6aecab747d169512d7150570f61ddfb0f1ee77c9

4.3.2 High-risk vulnerabilities

4.3.2.1 The available funds were not processed when the riskControl closed the position

In the Pool contract, when riskClose is triggered when the risk control liquidation is triggered, if the margin is insufficient and the pool order transfer fails, risk funds will be used to make up for the insufficient part, and all available funds of the user will be deducted. However, the user's available funds are not actually set to 0.

Fix suggestion: It is recommended that the available funds should be emptied after the transfer of insurance funds.

Code location: *Pool.sol

```
    if (lp1.moveLp1Fund(tempOrderID,tempProfit,moveProfit,tempNumber,tempPrice)){
        lpqAccount.amount =
lpqAccount.amount.sub(tempMaginAmount.add(tempMaginFee)).sub(lpqAccount.availableAmount);
        lpqAccount.availableAmount = 0;
        delete matchIds[tempOrderID];
        _safeTransferFrom(tokenAddress, riskFundAddr,address(lp1),moveRiskFund);
    }else{
        lpqAccount.amount = lpqAccount.amount.add(tempHoldFee);
        lpqAccount.availableAmount = lpqAccount.availableAmount.add(tempHoldFee);
        if(lpqAccount.availableAmount >= fixAmount){
            userProfit = tempProfit;
            lpqAccount.amount = lpqAccount.amount.sub(fixAmount);
            lpqAccount.availableAmount = lpqAccount.availableAmount.sub(fixAmount);
        }else{
                uint256 newFixAmount = fixAmount.sub(lpqAccount.availableAmount);
                if( getRiskFundAmount()   >= newFixAmount){
                    userProfit = tempProfit;
                     _safeTransferFrom(tokenAddress, riskFundAddr,address(this), newFixAmount);
                }else{
                        userProfit =
tempMaginAmount.add(tempMaginFee).add(lpqAccount.availableAmount).add(getRiskFundAmount());
                        _safeTransferFrom(tokenAddress, riskFundAddr,address(this), getRiskFundAmount());
                        }
              }
                   flag = true;
         }
```

Fix status: Fixed in this commit: 6aecab747d169512d7150570f61ddfb0f1ee77c9

4.3.3 Low-risk vulnerabilities

4.3.3.1 Insecure random number

In the Pool contract, the getMatchLp2Object function uses block difficulty and block time now as the random number seed to participate in the calculation of random numbers. But block difficulty and time can be predicted or manipulated.

Fix suggestion: It is recommended to use the random number provided by chainlink that cannot be manipulated.

Code location: DDS*Pool.sol

```
    function getMatchLp2Object(uint256 amount) public view returns(address){
        //create random indexe
        if(lpAddrs.length <1) {
            return address(0);
        }
        uint256 random = uint256(keccak256(abi.encodePacked(block.difficulty, now))).mod(lpAddrs.length);
        uint256 i;
        for (i = random;i < lpAddrs.length;i++){
             LP2Account memory  lp2Account = lpAccount[lpAddrs[i]];
            if(!lp2Account.isRejectOrder && lp2Account.availableAmount > amount){
                return lpAddrs[i];
            }
        }
        for (i = 0;i < random;i++){
            LP2Account memory  lp2Account = lpAccount[lpAddrs[i]];
            if(!lp2Account.isRejectOrder && lp2Account.availableAmount > amount){
                return lpAddrs[i];
            }
        }
    }
```

Fix status: No Fixed.

4.3.4 Enhancement Suggestions

4.3.4.1 Event missing

In the DDSContracts contract, the owner can set the key parameters of the contract through the setExchageAddress, setPoolTokenAddr, setPrivatePool, setPublicPool, setFormular, and setrepayFudAddr functions, but no event recording is performed.

Fix suggestion: In order to facilitate follow-up records and community viewing, it is recommended to record events for sensitive parameter modifications.

Code location: DDSContracts.sol

Fix status: Fixed in this commit: 6aecab747d169512d7150570f61ddfb0f1ee77c9

4.3.4.2 Does not follow the `Checks-effects-interactions` model

In Contracts and Pool2 contracts, when deposit and provide functions are used to recharge, the state is changed first, and then the corresponding tokens are transferred to the contract.

Fix suggestion: It is recommended to follow the Checks-effects-interactions model, first transfer the corresponding tokens and then change the state.

Code location: *Contracts.sol, *Pool2.sol

```
     function deposit(uint256 amount) public {
        AccountInfo   storage userAcc = userAccount[msg.sender];
        userAcc.depositAmount = userAcc.depositAmount.add(amount);
        userAcc.availableAmount = userAcc.availableAmount.add(amount);
         _safeTransferFrom(tokenAddr, msg.sender,address(this), amount);
         emit DDSDeposit(msg.sender, address(this), amount);
    }

    function provide(uint256 mintAmount) public {
         lastProvideTm[msg.sender] = block.timestamp;
         require(mintAmount >= minMint,"Mint Amount is too small");
         LP2Account memory lp2Acccount = lpAccount[msg.sender];
         lp2Acccount.amount = lp2Acccount.amount.add(mintAmount);
         lp2Acccount.availableAmount = lp2Acccount.availableAmount.add(mintAmount);
         lp2Acccount.holder = msg.sender;
         lpAccount[msg.sender] = lp2Acccount;
         //deposit to this contract address
         _safeTransferFrom(tokenAddress, msg.sender,address(this), mintAmount);

         if(!addressExist[msg.sender]){
             addressExist[msg.sender] = true;
             //addressIndex[msg.sender] = accIndex++;
             lpAddrs.push(msg.sender);
         }
         emit ProvideLP2(msg.sender, mintAmount);
    }
```

Fix status: Fixed in this commit: 6aecab747d169512d7150570f61ddfb0f1ee77c9

4.3.4.3 Redundant code

Provide price feed interfaces such as updateImpliedVolrate and updatePriceByOwner in the DDSFormular contract to update the price. However, these interfaces are not used by the Contracts contract, and these price-feeding interfaces have no permission control and can be called by any user.

Fix suggestion: If this interface is a test interface, it is recommended to remove it during formal deployment. If it will use the suggestions in subsequent iterations for permission control.

Code location: DDSFormular.sol

Fix status: No Fixed.

5. Audit Result

5.1 Conclusion

Audit Result : Low Risk

Audit Number : 0X002104060002

Audit Date : April 06, 2021

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist Team analysis tool audit of the codes for security issues. There are seven security issues found during the audit. There are two critical vulnerabilities, one high-risk vulnerabilities and one low-risk vulnerabilities. We also provide three enhancement suggestions. Since the random number issue has not been fixed yet, the contract still has low risk.

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.