Smart Contract Security Audit Report

Contents

1. Executive Summary

On Dec. 07, 2020, the SlowMist security team received the Aegis team's security audit application for LINA 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.

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

We audited the LINA Finance smart contract, the following is the related file information

Audit file information:

SHA256(linear-only-buildr.zip)

c213495ff245579fe42f9580c8ee85db716dc067872c538f009a390864bbdbc7

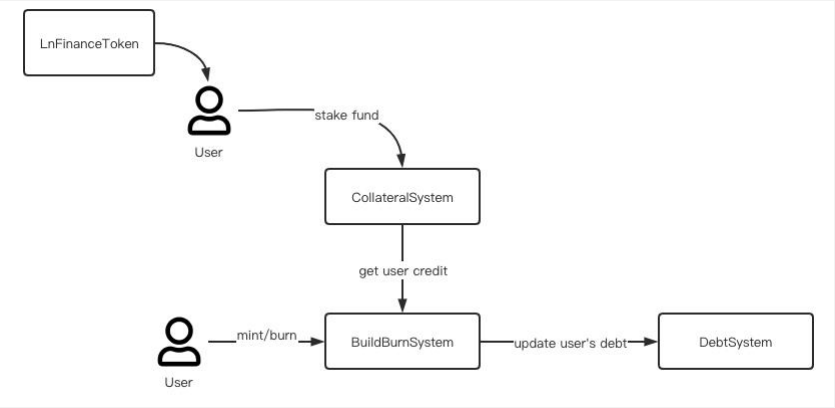
3.2 Project Structure

```
contracts
├── IAsset.sol
├── IERC20.sol
├── ISharePool.sol
├── LinearFinanceToken.sol
├── LnAccessControl.sol
├── LnAddressCache.sol
├── LnAdmin.sol
```

```
├── LnAsset.sol
├── LnAssetSystem.sol
├── LnBuildBurnSystem.sol
├── LnChainLinkPrices.sol
├── LnCollateralSystem.sol
├── LnConfig.sol
├── LnDebtSystem.sol
├── LnDefaultPrices.sol
├── LnEndAdmin.sol
├── LnErc20Handler.sol
├── LnFundVault.sol
├── LnLinearStaking.sol
├── LnObserver.sol
├── LnOperatorModifier.sol
├── LnPrices.sol
├── LnProxyERC20.sol
├── LnProxyImpl.sol
├── LnRewardLocker.sol
├── LnSharePool.sol
├── LnSimpleStaking.sol
├── LnTokenLocker.sol
├── LnTokenStorage.sol
├── LnTokenStorageLock.sol
├── Migrations.sol
└── SafeDecimalMath.sol
```

3.3 Contract Structure

The LINA Finance contract is mainly divided into three parts, the "LINA Finance" contract, the "DebtSystem" contract and the "Build Finance" contract. The "CollateralSystem " contract is used for the user's LINAToken mortgage. The DebtSystem contract records the user's debit and credit status. The BuildBurnSystem contract is used to generate and burn synthetic tokens. The overall structure of the project is as follows:



4. Code Overview

4.1 Contract Overview

We analyzed the function visibility of main contract, following are the result:

LnBuildBurnSystem			
Function	Visibility	Mutaibility	Modifier
constructor	Public	can modify state	LnAdmin
setPaused	External	can modify state	onlyAdmin
updateAddressCache	Public	can modify state	onlyAdmin
SetLusdTokenAddress	Public	can modify state	onlyAdmin
MaxCanBuildAsset	Public	-	-
BuildAsset	Public	can modify state	whenNotPaused
BuildMaxAsset	External	can modify state	whenNotPaused
_burnAsset	Internal	can modify state	-

BurnAsset	External	can modify state	whenNotPaused
BurnAssetToTarget	External	can modify state	whenNotPaused

LnAsset			
Function	Visibility	Mutaibility	Modifier
keyName	External	-	-
constructor	Public	-	LnErc20Handler
updateAddressCache	Public	-	onlyAdmin
_mint	Private	can modify state	-
mint	External	can modify state	OnlyIssueAssetRole
burn	External	can modify state	OnlyBurnAssetRole
_burn	Private	can modify state	-
setTotalSupplyAggre	External	can modify state	onlyObserver

LnErc20Handler			
Function	Visibility	Mutaibility	Modifier
constructor	Public	can modify state	LnAdmin LnProxyImpl
allowance	Public	-	-
balanceOf	External	-	-

setTokenStorage	External	can modify state	optionalProxy_onlyAdmin
_internalTransfer	Internal	can modify state	-
_transferByProxy	Internal	can modify state	-
_transferFromByProxy	Internal	can modify state	-
_beforeTokenTransfer	Internal	can modify state	-
transfer	External	can modify state	optionalProxy
transferFrom	External	can modify state	optionalProxy
approve	Public	can modify state	optionalProxy
addressToBytes32	Internal	-	-
emitTransfer	Internal	can modify state	-
emitApproval	Internal	can modify state	-
emitTokenStorageUpdated	Internal	can modify state	-

LnAssetSystem			
Function	Visibility	Mutaibility	Modifier
constructor	Public	can modify state	LnAddressStorage
addAsset	External	can modify state	onlyAdmin

removeAsset	External	can modify state	onlyAdmin
assetNumber	External	-	-
totalAssetsInUsd	Public	-	-
getAssetAddresses	External	-	-

LnDebtSystem			
Function	Visibility	Mutaibility	Modifier
constructor	Public	can modify state	LnAdmin
updateAddressCache	Public	can modify state	onlyAdmin
SetLastCloseFeePeriodAt	External	can modify state	OnlyDebtSystemRole
_pushDebtFactor	Private	can modify state	-
PushDebtFactor	External	can modify state	OnlyDebtSystemRole
_updateUserDebt	Private	can modify state	-
UpdateUserDebt	External	can modify state	OnlyDebtSystemRole
UpdateDebt	External	can modify state	OnlyDebtSystemRole
GetUserDebtData	External	-	-
_lastSystemDebtFactor	Private	-	-
LastSystemDebtFactor	External	-	-

GetUserCurrentDebtProportion	Public	-	-
GetUserDebtBalanceInUsd	External	-	-

LnCollateralSystem			
Function	Visibility	Mutaibility	Modifier
constructor	Public	can modify state	LnAdmin
setPaused	External	can modify state	onlyAdmin
updateAddressCache	Public	can modify state	onlyAdmin
updateTokenInfo	Private	can modify state	-
UpdateTokenInfo	External	can modify state	onlyAdmin
UpdateTokenInfos	External	can modify state	onlyAdmin
GetSystemTotalCollateralInUsd	Public	-	-
GetUserTotalCollateralInUsd	Public	-	-
GetUserCollateral	External	-	-
GetUserCollaterals	External	-	-

Collateral	External	can modify state	whenNotPaused
IsSatisfyTargetRatio	Public	-	-
MaxRedeemableInUsd	Public	-	-
MaxRedeemable	Public	-	-
RedeemMax	External	can modify state	whenNotPaused
_Redeem	Internal	can modify state	-
Redeem	Public	can modify state	whenNotPaused
constructor	External	payable	whenNotPaused
_CollateralEth	Internal	can modify state	-
CollateralEth	External	payable	whenNotPaused
RedeemETH	External	can modify state	whenNotPaused

4.2 Code Audit

4.2.1 Low-risk vulnerabilities

4.2.1.1 Inconsistent amount of token supply

TokenStorage is used to store token information, but there is no total supply to store tokens, resulting in the agent contract changing total_Supply may be inconsistent.

Location：lnAsset.sol Line 22

```
    constructor( bytes32 _key, address payable _proxy, LnTokenStorage _tokenStorage, string memory _name, string memory _symbol,
```

```
        uint _totalSupply, uint8 _decimals, address _admin) public
        LnErc20Handler(_proxy, _tokenStorage,_name,_symbol, _totalSupply, _decimals, _admin ) {

        mKeyName = _key;
        tokenStorage = _tokenStorage;
        name = _name;
        symbol = _symbol;
        totalSupply = _totalSupply;
        totalSupplyAggre = _totalSupply;
        decimals = _decimals;
    }
```

Fix suggestion：Store the total supply variable in the tokenStorage contract logic.

Fix status：Not repair. After communication with the project party, it is determined that when this contract first deployed, since totalSupply is 0, this problem will not be affected for the time being. This issue will be dealt with later when upgrading contracts.

4.2.1.2 Addresses may be out of sync between contracts

Lnbondalsystem, LnDebtSystem, and LnBuildBurn all implement the updateAddressCache function. When the address is updated, it is not guaranteed that each contract is synchronized with each other. It is recommended that the Address cache of one contract be updated at the same time as the address cache of other contracts.

Location：LnBuildBurnSystem.sol Line 46，LnCollateralSystem.sol Line 67 ，LnDebtSystem.sol Line 47

```
function updateAddressCache( LnAddressStorage _addressStorage ) onlyAdmin public override
    {
        priceGetter =      LnPrices( _addressStorage.getAddressWithRequire( "LnPrices",       "LnPrices address not valid" ) );
```

```
        debtSystem = LnDebtSystem( _addressStorage.getAddressWithRequire( "LnDebtSystem", "LnDebtSystem address not valid" ) );
        assetSys =  LnAssetSystem( _addressStorage.getAddressWithRequire( "LnAssetSystem","LnAssetSystem address not valid" ) );
        address payable collateralAddress = payable(_addressStorage.getAddressWithRequire( "LnCollateralSystem","LnCollateralSystem address not valid" ));
        collaterSys = LnCollateralSystem( collateralAddress );
        mConfig =          LnConfig( _addressStorage.getAddressWithRequire( "LnConfig",       "LnConfig address not valid" ) );

        emit updateCachedAddress( "LnPrices",               address(priceGetter) );
        emit updateCachedAddress( "LnDebtSystem",           address(debtSystem) );
        emit updateCachedAddress( "LnAssetSystem",          address(assetSys) );
        emit updateCachedAddress( "LnCollateralSystem", address(collaterSys) );
        emit updateCachedAddress( "LnConfig",               address(mConfig) );
    }
```

Fix suggestion：Create a contract to update the address, and uniformly update the contract address.

Fix status：Not epaired. After communication with the project party, the project party confirmed that the first deployment would not be affected. A later project might add a helper contract that calls several contracts simultaneously to ensure atomicity. Or you can pause the system and start upgrading again.

4.2.2 Enhancement Suggestions

4.2.1.2 LnBuildBurnSystem / LnCollateralSystem contract does not check the presence of the address

The LnCollateralSystem/LnCollateralSystem uses the updateAddressCache function to set the external contract address, but many places in the contract that use these addresses have be

en set without first checking the external address, which results in the failure of the contract function to be called.

Such as LnCollateralSystem/LnCollateralSystem GetSystemTotalCollateralInUsd GetUserTotalCollateralInUsd/GetUserCollateral function, did not check the corresponding machine address is already set, lead to function call.

```
    function GetSystemTotalCollateralInUsd() public view returns (uint256 rTotal) {
        for (uint256 i=0; i< tokenSymbol.length; i++) {
            bytes32 currency = tokenSymbol[i];
            if (tokenInfos[currency].totalCollateral > 0) { // this check for avoid calling getPrice when collateral is zero
                if (Currency_LINA == currency) {
                    uint256 totallina = tokenInfos[currency].totalCollateral.add(mRewardLocker.totalNeedToReward());
                    rTotal = rTotal.add( totallina.multiplyDecimal(priceGetter.getPrice(currency)) );
                } else {
                    rTotal =
rTotal.add( tokenInfos[currency].totalCollateral.multiplyDecimal(priceGetter.getPrice(currency)) );
                }
            }
        }

        if (address(this).balance > 0) {
            rTotal = rTotal.add(address(this).balance.multiplyDecimal(priceGetter.getPrice(Currency_ETH)));
        }
    }
```

Fix suggestion: After updateAddressCache sets a state variable initialize, set to true if the external address is set successfully, when we need to use the external address function, we need to check whether the initialize is true first, to avoid the call exception caused by the address is not set correctly.

Fix status: not fixed. After confirmation with the project side, the project side confirmed that the addresses were loaded correctly manually during the first deployment. This problem will be fixed in

subsequent iterations.

5. Audit Result

5.1 Conclusion

Audit Result : Some Risks

Audit Number : 0X002012150002

Audit Date : Dec. 15, 2020

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 two security issues found during the audit. There are two low-risk vulnerabilities. We also provide one enhancement suggestions. After communication and feedback with the Aegis 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