Smart Contract Security Audit Report

Contents

1. Executive Summary

On Jan. 18, 2020, the SlowMist security team received the LINA 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-audit-v1.tar.gz)

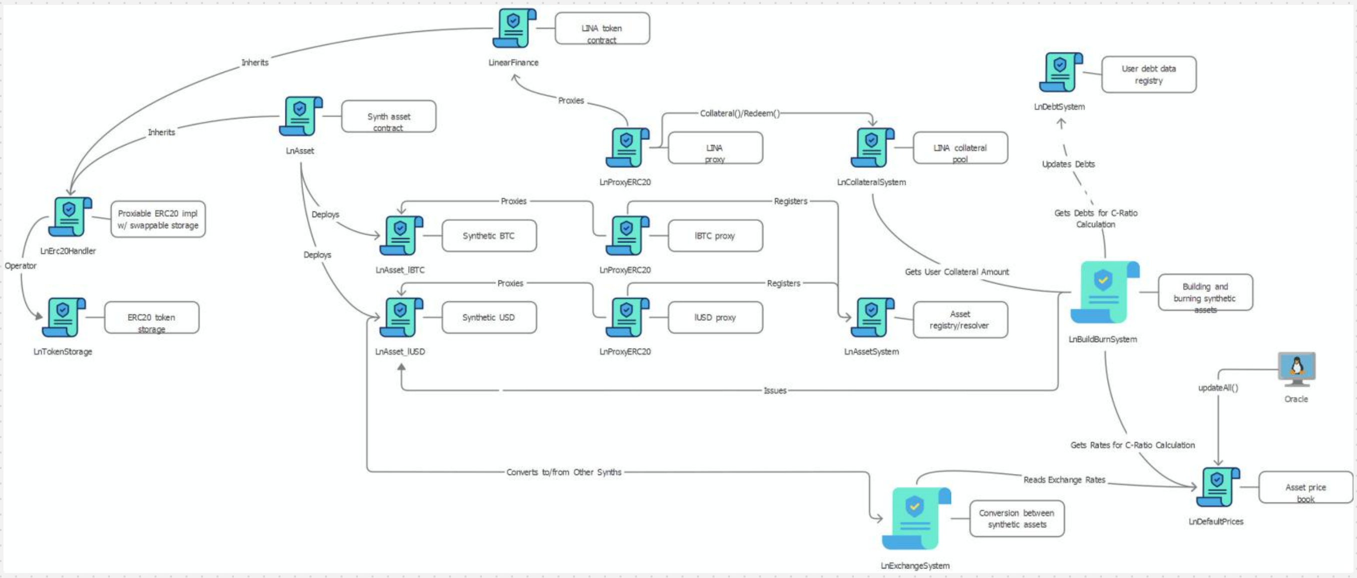
e248ec3aabad58dcfa6588a83a0e9c8937c7ae40550fad3e9f59dd1e6f05325d

3.2 Project Structure

```
contracts
├── LinearFinanceToken.sol
├── LnAccessControl.sol
├── LnAddressCache.sol
├── LnAddressStorage.sol
├── LnAdmin.sol
├── LnAssetSystem.sol
├── LnAssetUpgradeable.sol
├── LnBandProtocol.sol
├── LnBasePrices.sol
├── LnBuildBurnSystem.sol
├── LnCollateralSystem.sol
├── LnConfig.sol
├── LnDebtSystem.sol
├── LnDefaultPrices.sol
├── LnErc20Bridge.sol
├── LnExchangeSystem.sol
├── LnRewardLocker.sol
├── LnRewardSystem.sol
├── SafeDecimalMath.sol
├── interfaces
│   ├── IBandProtocolOracle.sol
│   ├── ILnAccessControl.sol
│   ├── ILnAddressStorage.sol
3 │   ├── ILnAsset.sol
│   ├── ILnAssetSystem.sol
│   ├── ILnCollateralSystem.sol
│   ├── ILnConfig.sol
│   ├── ILnDebtSystem.sol
│   ├── ILnPrices.sol
│   ├── ILnRewardLocker.sol
│   └── IMintBurnToken.sol
├── mock
│   ├── MockERC20.sol
│   └── MockLnRewardLocker.sol
└── upgradeable
└── LnAdminUpgradeable.sol
```

3.3 Contract Structure

The LINA Finance system is mainly divided into four parts, namely the CollateralSystem contract, the DebtSystem contract, the BuildBurnSystem contract and the ExchangeSystem contract. Among them, the CollateralSystem contract is used for users to pledge LINAToken. The DebtSystem contract records the user's loan status. The BuildBurnSystem contract is used to generate and burn synthetic tokens. ExchangeSystem is used to exchange other synthetic assets. 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:

LinearFinance			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LinearFinance_init	Public	can modify state	initializer
mint	External	can modify state	onlyAdmin

burn	External	can modify state	onlyAdmin

LnAdminUpgradeable			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnAdminUpgradeable_init	Public	can modify state	initializer
setCandidate	External	can modify state	onlyAdmin
becomeAdmin	External	can modify state	-

LnAccessControl			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnAccessControl_init	Public	can modify state	initializer
IsAdmin	Public	-	-
SetAdmin	Public	can modify state	-
SetRoles	External	can modify state	-
_setRoles	Private	can modify state	-
SetIssueAssetRole	Public	can modify state	-
SetBurnAssetRole	Public	can modify state	-
SetDebtSystemRole	Public	can modify state	-

LinearFinance			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LinearFinance_init	Public	can modify state	initializer
mint	External	can modify state	onlyAdmin
burn	External	can modify state	onlyAdmin

LnAddressStorage			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnAddressStorage_init	Public	can modify state	initializer
updateAll	External	can modify state	onlyAdmin
update	External	can modify state	onlyAdmin
getAddress	External	-	-
getAddressWithRequire	External	-	-

LnAddressStorage			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnAssetSystem_init	Public	can modify state	initializer

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

LnAssetSystem			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnAssetSystem_init	Public	can modify state	initializer
addAsset	External	can modify state	onlyAdmin
removeAsset	External	can modify state	onlyAdmin
assetNumber	External	-	-
totalAssetsInUsd	Public	-	-
getAssetAddresses	External	-	-

LnAssetUpgradeable			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnAssetUpgradeable_init	Public	can modify state	initializer

keyName	External	-	-
updateAddressCache	Public	can modify state	onlyAdmin
mint	External	can modify state	onlyIssueAssetRole
burn	External	can modify state	onlyBurnAssetRole

LnAssetUpgradeable			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnBandProtocol_init	Public	can modify state	initializer
addOracle	External	can modify state	onlyAdmin
removeOracle	External	can modify state	onlyAdmin
_getPriceData	Internal	-	-
removeFromArray	Private	can modify state	-

LnCollateralSystem			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnCollateralSystem_init	Public	can modify state	initializer
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	-	-
migrateCollateral	External	can modify state	onlyAdmin
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

LnBuildBurnSystem

Function Name	Visibility	Mutability	Modifiers
__LnBuildBurnSystem_init	Public	can modify state	initializer
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

LnBuildBurnSystem			
Function Name	Visibility	Mutability	Modifiers
__LnDebtSystem_init	Public	can modify state	initializer
updateAddressCache	Public	can modify state	onlyAdmin
SetLastCloseFeePeriodAt	External	can modify state	OnlyDebtSystemRole
importDebtData	External	can modify state	onlyAdmin

_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	-	-

LnBuildBurnSystem			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnExchangeSystem_init	Public	can modify state	initializer
updateAddressCache	Public	can modify state	onlyAdmin
exchange	External	can modify state	-
_exchange	Internal	can modify state	-

_addExchangeFee	Internal	can modify state	-

LnRewardLocker			
Function Name	**Visibility**	**Mutability**	**Modifiers**
__LnRewardLocker_init	Public	can modify state	initializer
setLinaAddress	External	can modify state	onlyAdmin
Init	External	can modify state	onlyAdmin
bulkAppendReward	External	can modify state	onlyAdmin
appendReward	External	can modify state	onlyFeeSys
Slimming	Public	can modify state	-
ClaimMaxable	Public	can modify state	-
_claim	Internal	can modify state	-
Claim	Public	can modify state	-

LnRewardSystem			
Function Name	**Visibility**	**Mutability**	**Modifiers**
getCurrentPeriodId	Public	-	-
getPeriodStartTime	Public	-	-

getPeriodEndTime	Public	-	-
__LnRewardSystem_init	Public	can modify state	initializer
setRewardSigner	External	can modify state	onlyAdmin
claimReward	External	can modify state	-
claimRewardFor	External	can modify state	-
_setRewardSigner	Private	can modify state	-
_claimReward	Private	can modify state	-

4.2 Code Audit

4.2.1 Low-risk vulnerabilities

4.2.1.1 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 updateAddressCache function

```
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.

4.2.1.2 User reward may not unlock in one time

The RewardLocker contract will call Slimming to clear the unlocked rewards when the user receives the reward. However, this function does not take into account the situation that all the user rewards are unlocked. When all the user rewards are unlocked, the user cannot get all the rewards through one claim.

Location: RewardLocker contract Slimming function

```
function Slimming(address _user) public {
        require(userRewards[_user].length > 1, "not data to slimming");
        RewardData storage claimable = userRewards[_user][0];
        for (uint256 i = 1; i < userRewards[_user].length; ) {
            if (now >= userRewards[_user][i].lockToTime) {
                claimable.amount = claimable.amount.add(userRewards[_user][i].amount);

                //swap last to current position
                uint256 len = userRewards[_user].length;
                userRewards[_user][i].lockToTime = userRewards[_user][len - 1].lockToTime;
                userRewards[_user][i].amount = userRewards[_user][len - 1].amount;
                userRewards[_user].pop(); // delete last one
            } else {
                i++;
            }
        }
```

Fix status: After communicating with the project party, it is confirmed that the unlocking period is at least half a year, and the risk here can be ignored.

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 been 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 : Low Risks

Audit Number : 0X002101260001

Audit Date : Jan. 26, 20201

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

SLOWMIST

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist