BLOCKLORDS Smart Contract

Audit Report

Mon Jan 08 2024

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

BLOCKLORDS Smart Contract Audit Report

1 Executive Summary

1.1 Project Information

Description	A token and lock contract
Type	Token
Auditors	ScaleBit
Timeline	Tue Dec 26 2023 - Wed Dec 27 2023
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/blocklords/smartcontracts
Commits	780a34b7311b5b6bcb8ac117cfc811559a82afdc ef87a3e311b4a826cdf0e5b89ba7531e1249eab9 dcda7a8eba6e36864c20bb1f1de288939fffae70

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
LRD	contracts/LRD/LRD.sol	1cbfc0d1a3a05441dca9545dc951866ec5d43c3f
LRDC	contracts/LRD/LRDClaim.sol	e6ddf3acc25de280fc6608d491f7734101a024b4
LRDL	contracts/LRD/LRDLock.sol	8c6254c98ea278dcf60d4298ca5d4e2256af96c6

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	9	9	0
Informational	2	2	0
Minor	4	4	0
Medium	0	0	0
Major	3	3	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by BLOCKLORDS to identify any potential issues and vulnerabilities in the source code of the LRD smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 9 issues of varying severity, listed below.

ID	Title	Severity	Status
LRD-1	Incorrect Amount Transfer in `exportLrd` Function	Major	Fixed
LRD-2	Incompatible With Deflationary Token	Major	Fixed
LRD-3	Lack of Events Emit	Minor	Fixed
LRD-4	Array Not Modified	Minor	Fixed
LRD1-1	Reentrancy Risk	Major	Fixed
LRD1-2	Lack of Constant Declaration	Minor	Fixed
LRD1-3	Redundant Initialization	Minor	Fixed
LRD1-4	Inaccurate Modifier Naming	Informational	Fixed
LRD1-5	Gas Optimization	Informational	Fixed

3 Participant Process

Here are the relevant actors with their respective abilities within the LRD Smart Contract:

Admin

- The Admin can mint `LRDS` tokens through `mintSeedRound()` / `mintStrategicRound()` / `mintPrivateRound()` / `mintGrowthRound()` / `mintCommunityRewards()` / `mintGameLaunchDrop()` / `mintFarmersBounty()` / `mintRulersBounty()` / `mintEmpireRewards()` / `mintDynastyIncentives()` / `mintLiquidity()` / `mintFoundationReserve()` / `mintAdvisors()` .
- The Admin can change the verifier through `changeVerifier()` .
- The Admin can add a verifier through `addVerifier()` .
- The Admin can change the bank address through `changeBank()` .
- The Admin can change the token address through `changeToken()` .
- The Admin can pause import tokens into the contract through `pauseGame()` .
- The Admin can resume import tokens into the contract through `resumeGame()` .
- The Admin can change the time for the specified type of import through `updateImportTypes()` .
- The Admin can add the import time type `addImportTypes()` .

Bridge

- The Bridge can mint `LRDS` tokens through `mint()` .
- The Bridge can burn the `LRDS` from the allowance address through `burnFrom()` .

User

- The User can claim `LRDS` tokens from a verifier through `exchangeLrd()` .
- The User can deposit `LRDS` tokens into the contract through `importLrd()` .
- The User can withdraw `LRDS` tokens from the contract through `exportLrd()` .

4 Findings

LRD-1 Incorrect Amount Transfer in `exportLrd` Function

Severity: Major

Status: Fixed

Code Location:

contracts/LRD/LRDLock.sol#112-117

Descriptions:

In the `exportLrd` function, the amount of the user to export is `_amount`, but the contract transfers all the tokens imported by the user, the same problem exists with the update of the variables `totalAmount` `params.amount` and so on.

Suggestion:

It is recommended to use the correct amount of export.

Resolution:

The client followed our suggestion and fixed this issue.

LRD-2 Incompatible With Deflationary Token

Severity: Major

Status: Fixed

Code Location:

contracts/LRD/LRDLock.sol#82-84

Descriptions:

In the `importLrd` function, due to the unknown address of `_token`, when the token is deflationary, the amount of tokens transferred to the contract by the user may not be accurate.

Suggestion:

It is recommended to add a check for the deflationary token as:

```
 amountBefore = _token.balanceOf(address(this));
_token.safeTransferFrom(msg.sender, address(this), _amount);
amountAfter = _token.balanceOf(address(this));
require(amountAfter - amountBefore >= amount);
```

Resolution:

The client followed our suggestion and fixed this issue.

LRD-3 Lack of Events Emit

Severity: Minor

Status: Fixed

Code Location:

contracts/LRD/LRDLock.sol#167-173

Descriptions:

The smart contract lacks appropriate events for monitoring sensitive operations, which could make it difficult to track sensitive actions or detect potential issues.

Suggestion:

It is recommended to emit events for those sensitive functions.

Resolution:

The client followed our suggestion and fixed this issue.

LRD-4 Array Not Modified

Severity: Minor

Status: Fixed

Code Location:

contracts/LRD/LRDLock.sol#94

Descriptions:

In the `exportLrd` function, the array `playerList` is not changed after withdraw tokens.

Suggestion:

It is recommended to remove `msg.sender` from the `playerlist`.

Resolution:

The client followed our suggestion and fixed this issue.

LRD1-1 Reentrancy Risk

Severity: Major

Status: Fixed

Code Location:

contracts/LRD/LRDClaim.sol#81;

contracts/LRD/LRDLock.sol#112

Descriptions:

In the `exchangeLrd` and `exportLrd` functions, due to the unknown token address, there may be a reentrancy risk if the token is callable during a transfer.

Suggestion:

It is recommended to add a no-reentrancy modifier.

Resolution:

The client followed our suggestion and fixed this issue.

LRD1-2 Lack of Constant Declaration

Severity: Minor

Status: Fixed

Code Location:

contracts/LRD/LRD.sol#16,17

Descriptions:

The variables `_million` and `_thousand` have not been modified in the contract, and can be optimized for gas consumption by using the `constant` declaration.

Suggestion:

It is recommended to add a `constant` declaration for the `_million` and `_thousand` variables.

Resolution:

The client followed our suggestion and fixed this issue.

LRD1-3 Redundant Initialization

Severity: Minor

Status: Fixed

Code Location:

contracts/LRD/LRD.sol#54

Descriptions:

The values in the `originalMints` array remain `false` until modified, so there is no need to reinitialize them as `false` in the `constructor`. This would consume additional gas.

Suggestion:

It is recommended to remove the initialization of the `originalMints` array in the `constructor`.

Resolution:

The client followed our suggestion and fixed this issue.

LRD1-4 Inaccurate Modifier Naming

Severity: Informational

Status: Fixed

Code Location:

contracts/LRD/LRD.sol#40

Descriptions:

The actual purpose of the `onlyMultisig` modifier is to check if the address is not a zero address, rather than verifying if the address is a multi-signature address. Therefore, the naming is ambiguous.

Suggestion:

It is recommended to modify the naming of the `onlyMultisig` modifier.

Resolution:

The client followed our suggestion and fixed this issue.

LRD1-5 Gas Optimization

Severity: Informational

Status: Fixed

Code Location:

contracts/LRD/LRDClaim.sol#66

Descriptions:

The code `params.statu != false` can be modified as `params.statu` to reduce gas consumption.

Suggestion:

It is recommended to modify the code as `params.statu`.

Resolution:

The client followed our suggestion and fixed this issue.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer