HAECHI AUDIT

Longdrink Finance

Smart Contract

Security Audit Report

June 28th, 2021

Overview

Project Summary

Target	Longdrink Finance
Platform	Binance Smart Chain / Solidity
Codebase	Github Repository (https://github.com/LongdrinkDefi/contracts)
Commit	4b36c0e7c2ceeccd59e6d1da6110048f1327dc81

Audit Summary

Delivery Date	June. 28, 2021
Author	Jasper Lee
Version	1.1

Vulnerability Summary

Severity	**# of Findings**
Critical	0
Major	0
Medium	0
Minor	1
Informational	2
Total	3

Potential Threats

There is following potential threats in Longdrink finance contract :

- Issues from using modules : Longdrink finance contract uses module such as Longdrink Mixer to mint and burn index tokens. there might be potential flaws in future version of modules or some consistency issues in changing module. threats from LongdrinkMixerV1 or adopted modules in future remains regardless of our audit.
- Pool drainage by flash loan : Longdrink finance contract uses PancakeSwap and BakerySwap to mint and burn, there might be possible pool drainage by flash loan attack to those Swaps. Since flash loan attack to swap contract is out of our scope, this threat remains regardless of our audit.
- Inoperability : The implementation may behave differently from the intended specifications. We test operability with full coverage unittest.

Key Findings

1. Unauthorized entity can call `Logic.mintExact()` Informational Acknowledged

Description

Longdrink finance uses `Logic.mintExact` to initialize the Longdrink token weights. this function is critical in contract operation, but unauthorized entity, who is not migrator or admin can call this function and ruin whole operation.

Logic.sol:L97-L112

```
97      function mintExact(
98          address[] memory _assets,
99          uint256[] memory _amountsIn,
100         uint256 _amountOut
101     ) public whenNotPaused {
102         require(totalSupply() == 0, "already initialized");
103         require(_assets.length == assets.length, "!asset-length");
104         require(_amountsIn.length == _assets.length, "!amounts-in");
105         require(_amountOut >= 1e6, "min-mint-1e6");
106
107         for (uint256 i = 0; i < _amountsIn.length; i++) {
108             ERC20(_assets[i]).safeTransferFrom(msg.sender, address(this), _amountsIn[i]);
109         }
110
111         _mint(msg.sender, _amountOut);
112     }
```

Recommendation

Logic.sol:L97-L112

```
97      function mintExact(
98          address[] memory _assets,
99          uint256[] memory _amountsIn,
100         uint256 _amountOut
101     ) public whenNotPaused authorized(MIGRATOR) {
102         require(totalSupply() == 0, "already initialized");
103         require(_assets.length == assets.length, "!asset-length");
104         require(_amountsIn.length == _assets.length, "!amounts-in");
105         require(_amountOut >= 1e6, "min-mint-1e6");
106
107         for (uint256 i = 0; i < _amountsIn.length; i++) {
108             ERC20(_assets[i]).safeTransferFrom(msg.sender, address(this), _amountsIn[i]);
109         }
110
111         _mint(msg.sender, _amountOut);
112     }
```

Please add modifier `authorized(MIGRATOR)` as above.

Acknowledgement

Longdrink Finance team confirmed this vulnerability. They determined that `mintExact` would occur approximately at the same time as the contract initialization, and no one could access it after the total supply was increased by mintExact, so there will be little threat from this vulnerability.

2. Possible inconsistency between `Logic` and `LongdrinkMixerV2.BEV`

Minor Acknowledged

Description

Longdrink finance uses `LongdrinkMixerV2` to mint and burn index token from BNB or BUSD. Logic can have `LongdrinkMixerV2` as its module state, and `LongdrinkMixerV2` can have Logic as BEV state. If Logic address and `LongdrinkMixerV2.BEV` differs, calling functions in `LongdrinkMixerV2` might results minting in unexpected contract.

Suppose There is 2 BEV tokens BEV1 and BEV2, and 2 Longdrink Mixer LongdrinkMixerV2-BEV1, LongdrinkMixerV2-BEV2 respectively. TIMELOCK can approve LongdrinkMixerV2-BEV2 as module of BEV1. If so, when someone calls its module with `BEV1.execute(LongdrinkMixerV2-BEV2, mintWithBnb)` , expecting minting BEV1 with BNB, it results minting BEV2, which is not expected, since LongdrinkMixerV2-BEV2 calls `BEV2.mint()`

Recommendation

Logic.sol:L200-L205

```
    function approveModule(address _module) public authorized(TIMELOCK) {
        require(address(IModule(_module).BEV) == address(this), "inconsistent");
        approvedModules[_module] = true;

        emit ModuleApproved(_module);
    }
```

Please check BEV state is same with Logic address when approve module.

Acknowledgement

Longdrink Finance team confirmed this vulnerability. and they will opt to alleviate this vulnerability in future implementation.

3. Mismatch between specifications in comment and actual implementation in `Logic.initialize()`

Informational Acknowledged

Description

The comment in `Logic.initialize()` states that _governance is authorized to set governance or market makers. But in the actual code, only market makers are given permission because `_setRoleAdmin()` can grant permission to only one address.

Logic.sol:L77-L88

```
77          // Governance can set governance OR market maker
78          _setRoleAdmin(GOVERNANCE, GOVERNANCE_ADMIN);
79          _setRoleAdmin(MARKET_MAKER, GOVERNANCE_ADMIN);
80          _setupRole(GOVERNANCE_ADMIN, _governance);
81          _setupRole(GOVERNANCE, _governance);
82
83          // Market maker admin
84          _setRoleAdmin(MARKET_MAKER, MARKET_MAKER_ADMIN);
85          for (uint256 i = 0; i < _marketMakers.length; i++) {
86              _setupRole(MARKET_MAKER_ADMIN, _marketMakers[i]);
87              _setupRole(MARKET_MAKER, _marketMakers[i]);
88          }
```

Recommendation

Replace Logic.sol:L79 with `_setupRole (MARKET_MAKER_ADMIN, _governance);`

Acknowledgement

Longdrink Finance team confirmed this vulnerability. and they will opt to alleviate this issue in future implementation.

Appendix A - Files in Scope

File	SHA-1 Hash
MasterChef.sol	f23ec774e6500a37b9b18ccdb12ab10b22678925
Constants.sol	f2a65a99823786af7da89b8212495663bb63ad28
IO.sol	5019ec7c3dcc766631cc1b38e11f9eacf02b7f42
Logic.sol	5dea43b5baaa654998ca03b071113dbe91a46110
Root.sol	4e5fd5b4b49a4dc2e60e9f883bcc4157da627e59
Storage.sol	51a4cd09e2280d2e903d4ad232e702047b4fc2c1
LongdrinkMixerV2.sol	65dbfd408fcff916ea8ffb65c6bc5e75cbd066b9
Helpers.sol	37c049cc541096ccb7f0ab356eb9969b820107d8

Appendix B - Test Results

```
Logic
  #initialize()
    ✓ Should fail if INITIALIZED flag is on
    ✓ Should set up ERC20 token properly
    ✓ Should set assets properly
    ✓ Should set setup fees properly
    ✓ Should set setup roles properly
    ✓ Should be timelock only allowed to set the migrator
    ✓ Should be governance able to set governance
    ✓ Should set market maker roles properly
  #mintExact()
    ✓ Should fail when paused
    ✓ Should fail when _assets length differ from assets length
    ✓ Should fail when _assets length differ from _amountsIn length
    ✓ Should fail when _amountsOut is smaller than 1,000,000
    ✓ Should mint properly
    ✓ Should fail when already initalized
  #getOne()
    ✓ Should get the amount of assets properly
  #getFees()
    ✓ Should get fees properly
  #unpause()
    ✓ Should fail when unpaused already
    ✓ Should unpause properly
  #setFee()
    ✓ Should fail when called by who is not timelock
    ✓ Should fail when mint fee is bigger than fee divisor
    ✓ Should fail when burn fee is bigger than fee divisor
    ✓ Should fail when fee recipient address is 0x0
    ✓ Should set fees properly
  #setAssets()
    ✓ Should fail when called by who is not timelock
    ✓ Should fail when paused
    ✓ Should set assets properly
  #rescueERC20()
    ✓ Should fail when called by who is not market maker or governance
    ✓ Should fail when rescue target is listed
    ✓ Should rescue unlisted token properly
  #approveModule()
    ✓ Should fail when called by who is not timelock
    ✓ Should approve module properly
  #revokeModule()
    ✓ Should fail when called by who is not timelock
    ✓ Should revoke module properly
  #execute()
    ✓ Should fail when called by who is not governance or timelock
    ✓ Should fail when called unapproved module
  #mint()
    ✓ Should fail when paused
    ✓ Should fail when totalSupply is 0
    ✓ Should charge fee when called by who is not market maker
    ✓ Should mint properly
  #viewMint()
    ✓ Should preview the assets and amount required to mint properly
  #burn()
    ✓ Should fail when pause
    ✓ Should fail when totalSupply is 0
    ✓ Should fail when burn amount is less than 1e6
    ✓ Should charge fee when called by who is not market maker
    ✓ Should burn properly
```

```
LongdrinkMixerV2
  #setGov()
    ✓ Should fail if called by non-governance account
    ✓ Should properly set governance
  #recoverERC20()
    ✓ Should fail if called by non-governance account
    ✓ Should properly recover unrelated token
  #mintWithBnb()
    ✓ Should properly mint with BNB
  #burnForBnb()
    ✓ Should properly burn with BNB
  #mintWithBusd()
    ✓ Should properly mint with BUSD
  #burnForBusd()
    ✓ Should properly burn with BUSD
```

File	% Stmts	% Branch	% Funcs	% Lines
Constants.sol	100	100	100	100
IO.sol	100	100	100	100
Logic.sol	100	100	100	100
Root.sol	100	100	100	100
Storage.sol	100	100	100	100
LongdrinkMixerV2.sol	100	100	100	100
Helpers.sol	100	100	100	100

Uncovered Lines

- MasterChef.sol : Since MasterChef is common library and had audited/verified by serveral experties. We did not write full unit test of MasterChef contract because we have audited focusing on other aspects rather than the inoperability of MasterChef contract.