Cybermonk Security

Smart Contract Audit Report

for

CHIPSHOP FINANCE

24th of May, 2021.

Document Properties

Client	chipshop.finance team
Title	Smart Contract Audit Report
Target	chipshop-contracts
Blockchain	Binance Smart Chain
Version	1.0
Author(s)	cybermonk
Auditor(s)	cybermonk
Classification	Public

Version Information

Version	**Date**	**Author(s)**	**Description**
1.0	June 2, 2021	cybermonk	Final release.
1.0-rc1	May 30, 2021	cybermonk	Status update and revision.
0.1	May 14, 2021	cybermonk	Initial draft and findings.

Contact

For more information about this document and its contents, please contact Cybermonk Security.

1. Preface

Given the opportunity to review the *chipshop.finance* documentation and solidity smart contract source code, we report our approach to evaluate potential security issues in the implementation, to prevent possible inconsistencies between smart contract code and architecture and to provide recommendations for further improvement. The audit review shows that the contracts could and were improved due to the presence of **high** to **low** impact security issues.

1.1 Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with several independent audits to ensure the security of smart contracts. Last but not least, this security audit should not be used as investment advice.

1.2 Overview

Chipshop Finance is an ambitious *"experimental regional economy"* project on the *Binance Smart Chain* (*BSC*). The aim of this project is to become the longest lasting and most successful stablecoin on the *BSC*, learning from the mistakes of others which went under-pegged.

The *Chipshop Finance* protocol is designed to expand and contract the supply of *CHIPS* tokens similar to the way central banks trade fiscal debt.

It is a multi-token protocol that consists of three tokens:

- *CHIPS*: the algorithmic token, pegged to *BETH*.
- *FISH*: shares, owned by holders which can be used to claim *CHIPS* inflation, when the network expands.
- *MPEAS*: *"mushy pea"* bonds, which can be purchased when the network is in contraction and can be redeemed for CHIPS when the network comes to its deflationary phase and the *CHIPS* price goes below the *BETH* price.

The project team is confident that the use of its novel *chip swap* mechanism will make it the first algorithmic stablecoin on the *BSC* to stay above its peg for a long time.

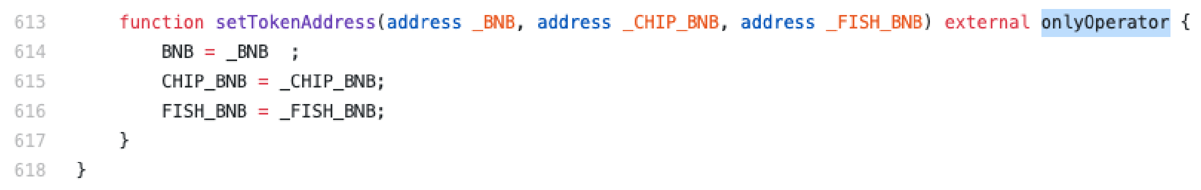
1.3 Audit Scope

During the audit process, we reviewed all the smart contracts in the project repository on the master branch. We were contracted to work in parallel with the development process at the beginning. Therefore we performed the audit in two phases. A mid one and a final one to sign the project go-live. The commit hash of the contract state which we last audited and signed off for the project go-live is fbe5f8a31cedb9f3f1366ce75bc12b02b0d286f9.

2. Code Review

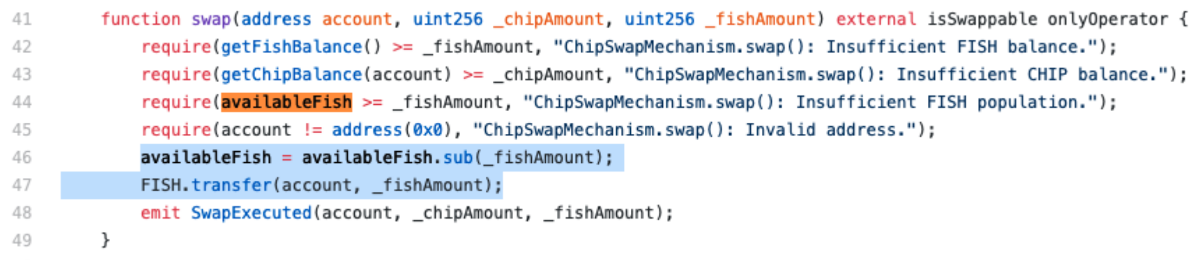
Cybermonk Security conducted a manual code review of all the smart contracts that are contained in the repository. The code of these contracts has been written according to the latest standards used within the Ethereum community and best practices of the Solidity community, making wise use of the OpenZeppelin contracts. The naming of variables is mostly logical and comprehensible, which results in the contract being useful to understand by a senior Solidity engineer. The code is however not documented at all and could have been more compact utilizing more elegant idioms. The team and the developers were available to us at all times to clear up any questions that we had.

During the middle phase of the project one ownership function modifier was missed resulting in a critical high-impact high-likelihood vulnerability capable of manipulating the *Treasury*'s knowledge on the amount of shares per coin. It was spotted and fixed immediately.



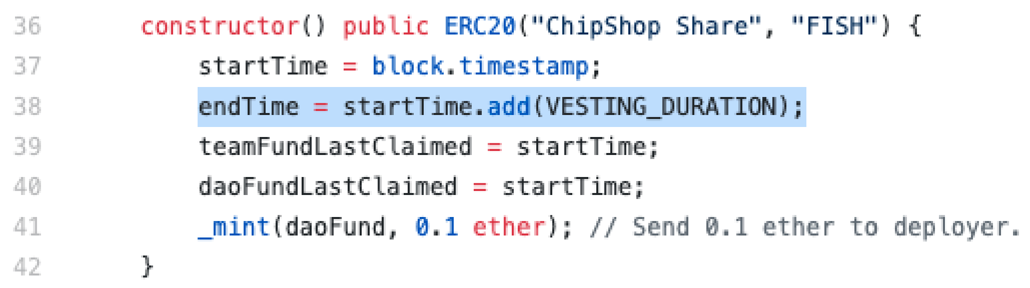
```
    function setTokenAddress(address _BNB, address _CHIP_BNB, address _FISH_BNB) external onlyOperator {
        BNB = _BNB  ;
        CHIP_BNB = _CHIP_BNB;
        FISH_BNB = _FISH_BNB;
    }
}
```

One medium-impact medium-likelihood reentrancy issue (SWC-107) was identified and fixed immediately within the *ChipSwap* mechanism.



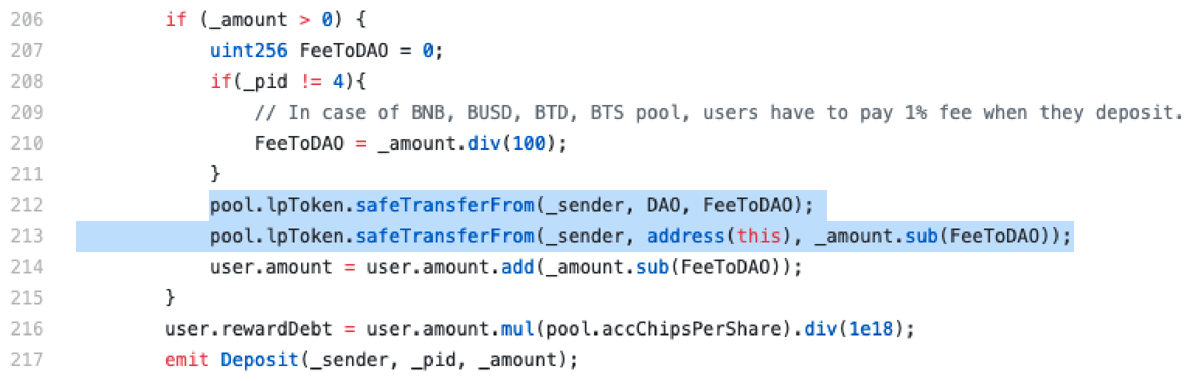
```
    function swap(address account, uint256 _chipAmount, uint256 _fishAmount) external isSwappable onlyOperator {
        require(getFishBalance() >= _fishAmount, "ChipSwapMechanism.swap(): Insufficient FISH balance.");
        require(getChipBalance(account) >= _chipAmount, "ChipSwapMechanism.swap(): Insufficient CHIP balance.");
        require(availableFish >= _fishAmount, "ChipSwapMechanism.swap(): Insufficient FISH population.");
        require(account != address(0x0), "ChipSwapMechanism.swap(): Invalid address.");
        availableFish = availableFish.sub(_fishAmount);
        FISH.transfer(account, _fishAmount);
        emit SwapExecuted(account, _chipAmount, _fishAmount);
    }
```

One integer overflow (SWC-101) of high-impact high-likelihood was taken care of as well within the *Fish Share*.



```
    constructor() public ERC20("ChipShop Share", "FISH") {
        startTime = block.timestamp;
        endTime = startTime.add(VESTING_DURATION);
        teamFundLastClaimed = startTime;
        daoFundLastClaimed = startTime;
        _mint(daoFund, 0.1 ether); // Send 0.1 ether to deployer.
    }
```

A gas spending optimization has also been proposed in the *Chips Reward Pool*, which could prevent unnecessary gas to be spent for 0 amount transfers.



```
        if (_amount > 0) {
            uint256 FeeToDAO = 0;
            if(_pid != 4){
                // In case of BNB, BUSD, BTD, BTS pool, users have to pay 1% fee when they deposit.
                FeeToDAO = _amount.div(100);
            }
            pool.lpToken.safeTransferFrom(_sender, DAO, FeeToDAO);
            pool.lpToken.safeTransferFrom(_sender, address(this), _amount.sub(FeeToDAO));
            user.amount = user.amount.add(_amount.sub(FeeToDAO));
        }
        user.rewardDebt = user.amount.mul(pool.accChipsPerShare).div(1e18);
        emit Deposit(_sender, _pid, _amount);
```

An additional double-check with two automated reviewing tools (one of them being the highest-paid version of MythX and one being Slither) did not find any additional bugs.

During our tests with methods like fuzz-testing, we were also not able to maliciously game the protocol nor to manipulate the Oracle.

The list of complete items which we checked for can be found in the Appendix.

3. Summary

Overall the smart contracts are well written and thought out. We like the idea of not recycling any foreign code in the core logic and writing it all by the team, since it enables them to understand every little detail and doesn't introduce any unnecessary risks. During the manual code review we found two alarming flaws and a medium one. Some code comments could have been present. We are confident that with the current state of the solidity smart contracts deployed to the mainnet, the *ChipShop Finance* team will enjoy cooking fish for a long time.

4. Appendix

The complete list of items we looked for during the audit process.

Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses

Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	Handling Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management

Additional Checks	Avoiding Use of Variadic Byte Array
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly