Security Audit Report for MancakeSwap

Date: May 28, 2024 **Version:** 1.1

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Mancake
Target	MancakeSwap

Version History

Version	Date	Description
1.0	May 07, 2024	First release
1.1	May 28, 2024	Second release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2024-06-19 13:11:48+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The focus of this audit is on the mancake-smart-contracts of Mancake [1]. While most smart contracts in the repository, which is a fork of PancakeSwap [2], are considered reliable in terms of both functionality and security, these files are not included in the scope of the audit. Please note that only changed files will be within the scope of our audit.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (Version 1), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
MancakeSwap	Version 1	7492b9765f30710151acc2ee2c637fcf33b99895
	Version 2	c9e8a97edd1d7c91da4229419826744833feacd0
	Version 3	d127f7998d04f0d3d0e2f5c7f142c479a49ab8a4

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the 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 independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] https://github.com/mancakeswap/mancake-smart-contracts

[2] https://github.com/pancakeswap

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

∗ Reentrancy
∗ DoS
∗ Access control
∗ Data handling and data flow
∗ Exception handling
∗ Untrusted external call and control flow
∗ Initialization consistency
∗ Events operation
∗ Error-prone randomness
∗ Improper use of the proxy system

1.3.2 DeFi Security

∗ Semantic consistency
∗ Functionality consistency
∗ Permission management
∗ Business logic
∗ Token operation
∗ Emergency mechanism
∗ Oracle security
∗ Whitelist and blacklist
∗ Economic impact
∗ Batch transfer

1.3.3 NFT Security

∗ Duplicated item
∗ Verification of the token receiver
∗ Off-chain metadata security

1.3.4 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [3] and Common Weakness Enumeration [4]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

		Likelihood	
		High	*Low*
Impact	*High*	High	Medium
	Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[3] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[4] https://cwe.mitre.org/

Chapter 2 Findings

In total, we have **two** recommendations. Besides, we also have **four** notes.

- Recommendation: 2
- Note: 4

ID	Severity	Description	Category	Status
1	-	Redundant code	Recommendation	Fixed
2	-	Lack of check in the Constructor()	Recommendation	Fixed
3	-	Forked logic from Pancake	Note	-
4	-	Potential centralization risks	Note	-
5	-	Zero farming speed of MasterChefV2	Note	-
6	-	Administrator can customize protocol fee rate	Note	-

The details are provided in the following sections.

2.1 Additional Recommendation

2.1.1 Redundant code

Status Fixed in Version 2

Introduced by Version 1

Description In the contract MancakeSwap, there is no logic of cake pool. Thus, the function migrateFromCakePool() is redundant.

```
function migrateFromCakePool() external nonReentrant {
    require(initialization, ’! initialized’);


    (uint256 shares, , , , , uint256 lockEndTime, uint256 userBoostedShare, , ) = CakePool.
        userInfo(msg.sender);


    require(lockEndTime > block.timestamp, ’Lock expired’);


    UserInfo storage user = userInfo[msg.sender];
    require(user.cakePoolType == 0, ’Already migrated’);


    user.cakePoolType = MIGRATION_FROM_CAKE_POOL_FLAG;
    uint256 totalShares = CakePool.totalShares();
    uint256 balanceOfCakePool = CakePool.balanceOf();
    // Subtract 1 is for precision round loss
    uint256 lockedCakeAmount = (shares * balanceOfCakePool) / totalShares - userBoostedShare -
        1;
    // will lock by proxy smart contract
    address proxy = ProxyForCakePoolFactory.deploy(msg.sender);
    isCakePoolProxy[proxy] = true;
    user.cakePoolProxy = proxy;
    user.migrationTime = uint48(block.timestamp);
    user.cakeAmount = uint128(lockedCakeAmount);
    user.lockEndTime = uint48(lockEndTime);


    IProxyForCakePool(proxy).createLockForProxy(lockedCakeAmount, lockEndTime);


    emit MigrateFromCakePool(msg.sender, proxy, lockedCakeAmount, lockE
```

Listing 2.1: VEMan.sol

Suggestion Remove the function `migrateFromCakePool()`.

2.1.2 Lack of check in the Constructor()

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `MancakeV3LmPool`, the `lastRewardTimestamp` is initialized in the `constructor()`. There is no check whether `rewardStartTimestamp` is greater than the current time.

```
constructor(address _pool, address _masterChef, uint32 rewardStartTimestamp) {
    pool = IPancakeV3Pool(_pool);
    masterChef = IMasterChefV3(_masterChef);
    lastRewardTimestamp = rewardStartTimestamp;
}
```

Listing 2.2: MancakeV3LmPool.sol

Suggestion Ensure `rewardStartTimestamp` is greater than `block.timestamp`.

2.2 Note

2.2.1 Forked logic from Pancake

Description Most of the contract in `MancakeSwap` is forked from `PancakeSwap`. Table below lists all the forked files in `MancakeaSwap`, which is out of our audit scope. Among all the files in the repository, only files `MasterChefV3.sol`, `MancakeV3LmPool.sol`, and `MancakeV3LmPoolDeployer.sol` are modified. The modified logic is covered in this audit.

	MancakeSwap	PancakeSwap
1	`projects/fee/contracts`	`https://github.com/pancakeswap/pancake-v3-contracts/tree/main/projects/masterchef-v3/contracts/receiver`

2	projects/masterchef-v3/contracts/Enumerable.sol	https://github.com/pancakeswap/pancake-v3-contracts/blob/main/projects/masterchef-v3/contracts/Enumerable.sol
3	projects/masterchef-v3/contracts/MancakeToken.sol	https://github.com/pancakeswap/pancake-smart-contracts/blob/master/projects/farms-pools/contracts/CakeToken.sol
4	projects/masterchef-v3/contracts/MasterChef.sol	https://github.com/pancakeswap/pancake-smart-contracts/blob/master/projects/farms-pools/contracts/MasterChef.sol
5	projects/masterchef-v3/contracts/MasterChefV2.sol	https://bscscan.com/address/0xa5f8C5Dbd5F286960b9d90548680aE5ebFf07652
6	projects/masterchef-v3/contracts/MasterChefV3.sol	https://github.com/pancakeswap/pancake-v3-contracts/blob/main/projects/masterchef-v3/contracts/MasterChefV3.sol
7	projects/masterchef-v3/contracts/SyrupBar.sol	https://github.com/pancakeswap/pancake-smart-contracts/blob/master/projects/farms-pools/contracts/SyrupBar.sol
8	projects/router/contracts	https://github.com/pancakeswap/pancake-v3-contracts/tree/main/projects/router/contracts
9	projects/stable-swap/contracts	https://github.com/pancakeswap/pancake-smart-contracts/tree/master/projects/stable-swap/contracts
10	projects/v3-core/contracts	https://github.com/pancakeswap/pancake-v3-contracts/tree/main/projects/v3-core/contracts
11	projects/v3-farm/contracts	https://github.com/pancakeswap/pancake-smart-contracts/tree/master/projects/vecake-farm-booster/v3/contracts
12	projects/v3-lm-pool/contracts	https://github.com/pancakeswap/pancake-v3-contracts/tree/main/projects/v3-lm-pool/contracts
13	projects/v3-periphery/contracts	https://github.com/pancakeswap/pancake-v3-contracts/tree/main/projects/v3-periphery/contracts
14	projects/voter/contracts	https://github.com/pancakeswap/pancake-smart-contracts/tree/master/projects/voter/contracts
15	projects/voter/contracts/RevenueSharingPool.sol	https://github.com/pancakeswap/pancake-smart-contracts/blob/master/projects/revenue-sharing-pool/v2/contracts/RevenueSharingPool.sol

16	projects/voter/contracts/RevenueSharingPoolFactory.sol	https://github.com/pancakeswap/pancake-smart-contracts/blob/master/projects/revenue-sharing-pool/v2/contracts/RevenueSharingPoolFactory.sol
17	projects/voter/contracts/VEMan.sol	https://github.com/pancakeswap/pancake-smart-contracts/blob/master/projects/vecake/contracts/VECake.sol

2.2.2 Potential centralization risks

Description Apart from the potential centralization risks in the forked logic of Pancake, there exist other potential centralization risks. Specifically, existing `CAKE allocation point` in the pool can be modified through the privileged function. If the `CAKE allocation point` is changed, the user's income can be reduced.

2.2.3 Zero farming speed of MasterChefV2

Description The value of the `cakeRateToRegularFarm` and `cakeRateToSpecialFarm` in the contract `MasterChefV2` is zero. The team guarantees that these parameters will be updated to reasonable values based on tokenomics at the project's launch to control the token emission.

2.2.4 Administrator can customize protocol fee rate

Description According to the protocol design, the `MancakeV3Pool` currently charges 100% of the swap fee as the protocol fee. Additionally, the privileged administrator has the ability to change the protocol fee rate.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS