verichains

SECURITY AUDIT OF

EASY CAKE TOKEN SMART CONTRACT

Public Report

Aug 26, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Aug 26, 2024. We would like to thank the EasyCake for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Easy Cake Token Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team identify some vulnerable issues in the contract code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Easy Cake Token Smart Contract

Easy Cake is a platform that has made a significant impact in the decentralized finance (DeFi) landscape. It offers users the ability to access various DeFi services with ease and transparency. Easy Cake is designed to be user-friendly, making it accessible for both beginners and experienced crypto investors.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Easy Cake Token Smart Contract. It was conducted on commit `fa2a1d9a1343109f8f48f13edcba6e01ce43af0a` from git repository link: *https://github.com/EasyCake-contracts/smart-contracts*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
`54b012b9484ec7813bbfd1578a6316d0556ff4d7559eaa0ebada675154f506bc`	`mCake.sol`

1.3. 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 RK87, our in-house smart contract security analysis tool.
- 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 were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy

- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

EasyCake acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. EasyCake understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, EasyCake agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the EasyCake will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the EasyCake, the final report will be considered fully accepted by the EasyCake without the signature.

2. AUDIT RESULT

2.1. Overview

The Easy Cake Token Smart Contract was developed using the `Solidity` language, with the required versions greater than `0.6.0` and less than `0.8.0`.

2.2. mCake Token

The mCakeToken is a customized ERC20 token that extends the functionality of a governance token, similar to PancakeSwap. Through its governance features, users can leverage their token balance to vote on various features within the EasyCake platform. The contract is built upon the `Ownable` extension, designating the deployer as the default owner of the contract. Ownership can be transferred to another address by the owner. Additionally, the owner has the capability to lock users' balances for a specified period through the lock function as part of the project's strategic management.

2.3. Findings

During the audit process, the audit team identify some vulnerable issues in the contract code.

2.3.1. mCake.sol - Lose _locks data when self transfer in `transferAll` function HIGH

The `_locks` data is used to record the users' locked in contract and user can withdraw completely after the lock period. However, the `_locks` data will be lost when the user self transfer in `transferAll` function. This will cause the user to lose the locked data and cannot withdraw it after the lock period.

```
function transferAll(address _to) public {
        _locks[_to] = _locks[_to].add(_locks[msg.sender]);

        if (_lastUnlockBlock[_to] < lockFromBlock) {
            _lastUnlockBlock[_to] = lockFromBlock;
        }

        if (_lastUnlockBlock[_to] < _lastUnlockBlock[msg.sender]) {
            _lastUnlockBlock[_to] = _lastUnlockBlock[msg.sender];
        }

        _locks[msg.sender] = 0; // <-- _locks data will be lost when self transfer
(to==msg.sender)
        _lastUnlockBlock[msg.sender] = 0;

        _transfer(msg.sender, _to, balanceOf(msg.sender));
    }
```

UPDATES

- *Aug 28, 2024*: This issue has been acknowledged and fixed by the EasyCake team.

2.3.2. mCake.sol - Operator can manualMint more than the manualMintLimit MEDIUM

The `manualMint` function allows operators to mint tokens with a manualMintLimit. However, the if-statement in this function is incorrect, allowing the operator to mint more than the `manualMintLimit`.

```
function manualMint(address _to, uint256 _amount) public onlyAuthorized {
    if(manualMinted < manualMintLimit){
        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);
        manualMinted = manualMinted.add(_amount);
    }
}
```

RECOMMENDATION

Add `require(manualMinted + _amount <= manualMintLimit)` to prevent operator mint more than the manualMintLimit.

UPDATES

- *Aug 28, 2024*: This issue has been acknowledged and fixed by the EasyCake team.

2.3.3. mCake.sol - Centralize Mechainism INFORMATIVE

The contract currently relies on a centralized mechanism where the owner and authorized members control the lock and releaseTime logic. This centralization creates a potential vulnerability, as a compromise of these accounts could allow malicious actors to exploit the system.

```
function lock(address _holder, uint256 _amount) public onlyOwner {
        require(_holder != address(0), "ERC20: lock to the zero address");
        require(_amount <= balanceOf(_holder), "ERC20: lock amount over blance");

        _transfer(_holder, address(this), _amount);

        _locks[_holder] = _locks[_holder].add(_amount);
        _totalLock = _totalLock.add(_amount);
        if (_lastUnlockBlock[_holder] < lockFromBlock) {
            _lastUnlockBlock[_holder] = lockFromBlock;
        }
        emit Lock(_holder, _amount);
    }
```

UPDATES

- *Aug 28, 2024*: This issue has been acknowledged by the EasyCake team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Aug 26, 2024*	Public Report	Verichains Lab

Table 2. Report versions history