Security Audit Report for CakePie Contracts

Date: November 30, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2023.11.30 17:24:31
+08'00'

Contents

Report Manifest

Item	Description
Client	Magpie XYZ
Target	CakePie Contracts

Version History

Version	Date	Description
1.0	November 30, 2023	First Release

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 5 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 target of this audit is the code repositiory of CakePie Contracts[1] of Magpie XYZ. The CakePie Contracts runs a CakeRush campaign that users can convert their CAKE token or locked CAKE position from PancakeSwap on CakePie. Please note that only `CakeRush.sol` and `PancakeStakingBNBChain.sol` are included in the audit scope, while other files are out of scope for this 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
CakePie Contracts	`Version 1`	`b0bafb5e061caa0583ce0a87d06d93e83e3aa66a`
	`Version 2`	`8246b8cc60a15bd7a88e74402c8cb7d13052df4f`
	`Version 3`	`7c01361d199966611a97a0e31562505b9f39cf3c`

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/magpiexyz/cakepie_contract/tree/1stAudit

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 [2] and Common Weakness Enumeration [3]. 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

Impact \ Likelihood	*High*	*Low*
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.

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

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

Chapter 2 Findings

In total, we find **two** potential issues. Besides, we also have **three** recommendations and **one** note.

- High Risk: 1
- Low Risk: 1
- Recommendation: 3
- Note: 1

ID	Severity	Description	Category	Status
1	Low	Potential inconsistent state after parameter reset	Software Security	Fixed
2	High	Repeated claim of mCake rewards	Software Security	Fixed
3	-	Check the parameters in initialization functions	Recommendation	Acknowledged
4	-	Check the parameters in `CakeRush` contracts	Recommendation	Fixed
5	-	Extra conditions in modifiers	Recommendation	Acknowledged
6	-	Potential centralization risk	Note	-

The details are provided in the following sections.

2.1 Software Security

2.1.1 Potential inconsistent state after parameter reset

Severity Low

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `CakeRush` contract distributes rewards according to several parameters. The following functions allows the project maintainer to reset some of the parameters:

```
324    function resetMultiplier() external onlyOwner {
325        uint256 len = rewardMultiplier.length;
326        for (uint8 i = 0; i < len; ++i) {
327            rewardMultiplier.pop();
328            rewardTier.pop();
329        }
330
331        tierLength = 0;
332    }
333
334    function resetTimeWeighting() external onlyOwner {
335        uint256 len = weightedTime.length;
336        for (uint8 i = 0; i < len; ++i) {
337            weightedTime.pop();
338            weighting.pop();
339        }
340
341        weightLength = 0;
```

```
342     }
```

Listing 2.1: CakeRush.sol

However, these functions only reset the parameters, but not the user information stored in the `userInfos` state variable. As a result, calculations in the `CakeRush` contract can fail due to the inconsistent state. For example, if the parameters are reset and set to incorrect values, the subtraction on Line 155 can fail due to integer underflow.

```
128     function quoteConvert(
129         uint256 _amountToConvert,
130         address _account
131     )
132         external
133         view
134         returns (
135             uint256 newUserFactor,
136             uint256 newTotalFactor,
137             uint256 newUserWeightedFactor,
138             uint256 newWeightedTotalFactor
139         )
140     {
141         if (_amountToConvert == 0 || rewardMultiplier.length == 0 || weighting.length == 0)
142             return (0, 0, 0, 0);
143 
144         UserInfo storage userInfo = userInfos[_account];
145         uint256 accumulated = _amountToConvert + userInfo.converted;
146 
147         uint256 factorAccuNoWeighting = 0;
148         uint256 i = 1;
149         while (i < rewardTier.length && accumulated > rewardTier[i]) {
150             factorAccuNoWeighting += (rewardTier[i] - rewardTier[i - 1]) * rewardMultiplier[i - 1];
151             i++;
152         }
153         factorAccuNoWeighting += (accumulated - rewardTier[i - 1]) * rewardMultiplier[i - 1];
154 
155         uint256 factorToEarnNoWeighting = (factorAccuNoWeighting / DENOMINATOR) - userInfo.factor;
```

Listing 2.2: CakeRush.sol

What's worse, if users call `convert` or `convertWithCakePool` immediately after resetting the parameters (before the new parameters are set, for example through back-running) can lead to the reset of the total and weighted factors recorded inside the contract, due to the logic on Line 141-142.

Impact Resetting parameters can lead to inconsistent and incorrect state.

Suggestion Set the new parameters after clearing the old ones.

Feedback from the Project Multipliers wont be reset once the cake rush campaign starts.

2.1.2 Repeated claim of mCake rewards

Severity High

Status Fixed in `Version 3`

Introduced by Version 2

Description After locking CAKE tokens in the CakeRush contract, users can claim mCake tokens as rewards through the claim function. However, the claim function contains a issue that allows the users to claim the rewards multiple times. In the following code segment, if the user.converted amount is larger than the claimedMCake of the user, claim would transfer or deposit a total amount of user.converted to the user. The correct implementation should only return user.converted - claimedMCake[user], so the current implementation effectively allows a user to repeatedly claim the mCake rewards.

```
269    function claim(bool _isStake) external nonReentrant {
270        UserInfo storage userInfo = userInfos[msg.sender];
271        if (claimedMCake[msg.sender] >= userInfo.converted) revert AlreadyClaimed();
272        if (_isStake && userInfo.converted > 0) {
273            if (masterCakepie == address(0)) revert MasterCakepieNotSet();
274            IERC20(mCakeOFT).safeApprove(address(masterCakepie), userInfo.converted);
275            IMasterCakepie(masterCakepie).depositFor(
276                address(mCakeOFT),
277                address(msg.sender),
278                userInfo.converted
279            );
280        } else if (userInfo.converted > 0) {
281            IERC20(mCakeOFT).transfer(msg.sender, userInfo.converted);
282            emit Claim(msg.sender, userInfo.converted);
283        }
284
285        claimedMCake[msg.sender] = userInfo.converted;
286    }
```

Listing 2.3: CakeRush.sol

Impact Users are able to repeatedly claim mCake rewards.

Suggestion Revise the reward claim logic.

2.2 Additional Recommendation

2.2.1 Check the parameters in initialization functions

Status Acknowledged

Introduced by Version 1

Description In the initialization functions of the CakeRush and PancakeStakingBNBChain contracts, there are parameters that cannot be changed after initialization. It is recommended that these parameters should be checked in the initialization functions.

```
105    function __CakeRush_init(
106        address _cake,
107        address _mCakeOFT,
108        address _masterCakepie
109    ) public initializer {
110        __Ownable_init();
111        __ReentrancyGuard_init();
112        __Pausable_init();
```

```
113         cake = _cake;
114         mCakeOFT = _mCakeOFT;
115         masterCakepie = _masterCakepie;
116     }
```

Listing 2.4: CakeRush.sol

Impact N/A

Suggestion Check parameters in initialization functions.

2.2.2 Check the parameters in `CakeRush` contracts

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the `CakeRush` contract, several parameters regarding the reward distribution can be added. However, there is no check that these parameters are set correctly according to the assumptions in the contract. Specifically, in `setMultipler` and `setTimeWeighting` function, extra conditions must be checked (i.e. the monotonic increasing property of the `rewardTier` and `weightedTime` array).

```
294     function setMultiplier(
295         uint256[] calldata _multiplier,
296         uint256[] calldata _tier
297     ) external onlyOwner {
298         if (_multiplier.length == 0 || (_multiplier.length != _tier.length)) revert LengthInvalid()
                ;
299
300         for (uint8 i = 0; i < _multiplier.length; ++i) {
301             if (_multiplier[i] == 0) revert InvalidAmount();
302             rewardMultiplier.push(_multiplier[i]);
303             rewardTier.push(_tier[i]);
304             tierLength += 1;
305         }
306     }
```

Listing 2.5: CakeRush.sol

Impact N/A

Suggestion Check parameters in the functions that set the parameters.

2.2.3 Extra conditions in modifiers

Status Acknowledged

Introduced by `Version 1`

Description In the `CakeRush` contract, the `_onlyPancakeStaking` modifier has an extraneous condition. According to the semantics of this modifier, checking `msg.sender != pancakeStaking` would be sufficient.

```
120     modifier _onlyPancakeStaking() {
121         if (pancakeStaking == address(0) || msg.sender != pancakeStaking)
122             revert OnlyPancakeStaking();
123         _;
```

```
124     }
```

Listing 2.6: CakeRush.sol

Impact N/A

Suggestion Remove extraneous conditions in the modifier.

2.3 Note

2.3.1 Potential centralization risk

Description The owner of `CakeRush` holds significant privileges to modify critical configurations. This creates a single point of failure. If an attacker were to compromise the owner, they could potentially incapacitate the entire system.

Additionally, the CAKE tokens in the contract are not explicitly handled to lock them into the VECake contract. Instead, `CakeRush` allows the owner to withdraw all those CAKEs, which means the owner must lock the CAKE tokens after withdrawing. However, the logic is not guaranteed at the code level, which also brings centralization concerns.

Feedback from the Project The team make the owner as a multisig to mitigate the risk.