SMART CONTRACT AUDIT REPORT

for

PancakeSwap Prediction V2

Prepared By: Yiqun Chen

Hangzhou, China

August 20, 2021

Document Properties

Client	PancakeSwap
Title	Smart Contract Audit Report
Target	PancakeSwap Prediction V2
Version	1.0
Author	Shulin Bie
Auditors	Shulin Bie, Xuxian Jiang
Reviewed by	Yiqun Chen
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	August 20, 2021	Shulin Bie	Final Release
1.0-rc	August 19, 2021	Shulin Bie	Release Candidate

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Yiqun Chen
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the PancakeSwap Prediction V2 protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About PancakeSwap Prediction V2

PancakeSwap is the leading decentralized exchange on Binance Smart Chain, with very high trading volumes in the market. The PancakeSwap Prediction V2 protocol is one of the core functions of PancakeSwap, which is designed as a decentralized BNB price prediction platform. It allows the user to profit from the BNB price rises and falls. The PancakeSwap Prediction V2 protocol enriches the PancakeSwap ecosystem and also presents a unique contribution to current DeFi ecosystem.

The basic information of PancakeSwap Prediction V2 is as follows:

Table 1.1: Basic Information of PancakeSwap Prediction V2

Item	Description
Target	PancakeSwap Prediction V2
Website	https://pancakeswap.finance/
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	August 20, 2021

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/pancakeswap/pancake-contracts/tree/master/projects/predictions/v2 (4c3c76d)

And this is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/pancakeswap/pancake-contracts/tree/master/projects/predictions/v2 (c564432)

1.2 About PeckShield

PeckShield Inc. [7] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [6]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic 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
Semantic Consistency Checks	Semantic Consistency Checks
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
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [5], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the `PancakeSwap Prediction V2` implementation. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logic, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	0	
Low	1	■
Informational	2	■ ■
Undetermined	0	
Total	3	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 low-severity vulnerability, and 2 informational recommendations.

Table 2.1: Key PancakeSwap Prediction V2 Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Trust Issue Of Admin Keys	Security Features	Confirmed
PVE-002	Informational	Improved Gas Efficiency In betBear()/betBull()	Coding Practices	Fixed
PVE-003	Informational	Redundant State/Code Removal	Coding Practices	Fixed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Trust Issue Of Admin Keys

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: PancakePredictionV2
- Category: Security Features [3]
- CWE subcategory: CWE-287 [1]

Description

In the PancakeSwap Prediction V2 protocol, there is a privileged account that plays a critical role in governing and regulating the protocol-wide operations (e.g., configuring various system parameters). In the following, we show the representative functions potentially affected by the privilege of the account.

```
    /**
     * @notice Set Oracle address
     * @dev Callable by admin
     */
    function setOracle(address _oracle) external whenPaused onlyAdmin {
        require(_oracle != address(0), "Cannot be zero address");
        oracleLatestRoundId = 0;
        oracle = AggregatorV3Interface(_oracle);

        // Dummy check to make sure the interface implements this function properly
        oracle.latestRoundData();

        emit NewOracle(_oracle);
    }

    ...

    /**
     * @notice Set reward and treasury rates
     * @dev Callable by admin
     */
```

```
    function setRewardAndTreasuryRates(uint256 _rewardRate, uint256 _treasuryRate)
        external whenPaused onlyAdmin {
        require(_rewardRate + _treasuryRate == 10000, "Must equal 10000 (100 * 1e2)");
        rewardRate = _rewardRate;
        treasuryRate = _treasuryRate;

        emit NewRewardAndTreasuryRates(currentEpoch, rewardRate, treasuryRate);
    }
```

Listing 3.1: `PancakePredictionV2::setOracle()&&setRewardAndTreasuryRates()`

We emphasize that the privilege assignment may be necessary and consistent with the protocol design. However, it is worrisome if the privileged account is not governed by a `DAO`-like structure. Note that a compromised account would allow the attacker to modify a number of sensitive system parameters, which directly undermines the assumption of the `PancakeSwap Prediction V2` design.

Recommendation Promptly transfer the privileged account to the intended `DAO`-like governance contract. All changed to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been confirmed by the team. The privileged account will be managed by a multi-sig account.

3.2 Improved Gas Efficiency In betBear()/betBull()

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `PancakePredictionV2`
- Category: Coding Practices [4]
- CWE subcategory: CWE-287 [1]

Description

In the `PancakeSwap Prediction V2` protocol, the `betBear()` function is designed to bet bear position by the user. While examining the logic of the `betBear()` function, we notice the `currentEpoch` storage variable that indicates the current active prediction round is read repeatedly in the function, which lends to unnecessary gas cost.

To elaborate, we show below the related code snippet of the `betBear()` function. In the `betBear()` function, the `require(epoch == currentEpoch, "Bet is too early/late")` is called to make sure the user can only bet bear position for the current active prediction round specified by the `currentEpoch` storage variable. In other words, the `epoch` parameter will be equal to the `currentEpoch` storage variable, otherwise the transaction will be reverted. In the subsequent implementation of the `betBear()`function, we observe the `currentEpoch` storage variable is read repeatedly. Since the `currentEpoch` storage variable is equal to the input `epoch` parameter, we suggest to replace `currentEpoch` with `epoch` to improve gas efficiency.

```
    function betBear(uint256 epoch) external payable whenNotPaused nonReentrant notContract {
        require(epoch == currentEpoch, "Bet is too early/late");
        require(_bettable(currentEpoch), "Round not bettable");
        require(msg.value >= minBetAmount, "Bet amount must be greater than minBetAmount");
        require(ledger[currentEpoch][msg.sender].amount == 0, "Can only bet once per round");

        // Update round data
        uint256 amount = msg.value;
        Round storage round = rounds[currentEpoch];
        round.totalAmount = round.totalAmount + amount;
        round.bearAmount = round.bearAmount + amount;

        // Update user data
        BetInfo storage betInfo = ledger[currentEpoch][msg.sender];
        betInfo.position = Position.Bear;
        betInfo.amount = amount;
        userRounds[msg.sender].push(currentEpoch);

        emit BetBear(msg.sender, currentEpoch, amount);
    }
```

Listing 3.2: `PancakePredictionV2::betBear()`

Note the `betBull()` routine can be similarly improved.

Recommendation Replace the usage of the `currentEpoch` storage variable with the input `epoch` parameter in the `betBear()/betBull()` routines.

Status The issue has been addressed by the following commit: `c564432`.

3.3 Redundant State/Code Removal

- ID: PVE-003
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: PancakePredictionV2
- Category: Coding Practices [4]
- CWE subcategory: CWE-563 [2]

Description

In the PancakeSwap Prediction V2 protocol, the genesisLockRound() function is designed to lock the genesis prediction round. After the prediction round is locked, the transactions for the prediction round will be denied. According to the current design, the prediction round should be locked between the start lock time specified by the lockTimestamp and the end lock time specified by the lockTimestamp plus the bufferSeconds. Once the prediction round is not locked in the given period of time, the prediction round will never be locked. While examining the logic of the genesisLockRound() function, we notice there are some redundant codes that can be safely removed.

To elaborate, we show below the related code snippet of the PancakePredictionV2 contract. In the genesisLockRound() function, the require(block.timestamp <= rounds[currentEpoch].lockTimestamp + bufferSeconds, "Can only lock round within bufferSeconds") is called (line 579 - line 582) to ensure the prediction round can only be locked in the given period of time. It comes to our attention that there is the same protection logic in the _safeLockRound() function, which will be subsequently called (line 281) in the genesisLockRound() function. We may intend to remove the redundant protection (line 579 - line 582) in the genesisLockRound() function.

```
269     function genesisLockRound() external whenNotPaused onlyOperator {
270         require(genesisStartOnce, "Can only run after genesisStartRound is triggered");
271         require(!genesisLockOnce, "Can only run genesisLockRound once");
272         require(
273             block.timestamp <= rounds[currentEpoch].lockTimestamp + bufferSeconds,
274             "Can only lock round within bufferSeconds"
275         );
276
277         (uint80 currentRoundId, int256 currentPrice) = _getPriceFromOracle();
278
279         oracleLatestRoundId = uint256(currentRoundId);
280
281         _safeLockRound(currentEpoch, currentRoundId, currentPrice);
282
283         currentEpoch = currentEpoch + 1;
284         _startRound(currentEpoch);
285         genesisLockOnce = true;
286     }
287
```

```
    ...

    function _safeLockRound(
        uint256 epoch,
        uint256 roundId,
        int256 price
    ) internal {
        require(rounds[epoch].startTimestamp != 0, "Can only lock round after round has
            started");
        require(block.timestamp >= rounds[epoch].lockTimestamp, "Can only lock round
            after lockTimestamp");
        require(
            block.timestamp <= rounds[epoch].lockTimestamp + bufferSeconds,
            "Can only lock round within bufferSeconds"
        );
        Round storage round = rounds[epoch];
        round.closeTimestamp = block.timestamp + intervalSeconds;
        round.lockPrice = price;
        round.lockOracleId = roundId;

        emit LockRound(epoch, roundId, round.lockPrice);
    }
```

Listing 3.3: `PancakePredictionV2::genesisLockRound()&&_safeLockRound()`

Recommendation Consider the removal of the redundant code.

Status The issue has been addressed by the following commit: `c564432`.

4 Conclusion

In this audit, we have analyzed the `PancakeSwap Prediction V2` design and implementation. `PancakeSwap` is the leading decentralized exchange on Binance Smart Chain, with very high trading volumes in the market. The `PancakeSwap Prediction V2` protocol is one of the core functions of `PancakeSwap`, which is designed as a decentralized BNB price prediction platform. It allows the user to profit from the BNB price rises and falls. The `PancakeSwap Prediction V2` protocol enriches the `PancakeSwap` ecosystem and also presents a unique contribution to current DeFi ecosystem. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[2] MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

[3] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[4] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[5] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[6] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[7] PeckShield. PeckShield Inc. https://www.peckshield.com.