# BNBPot Roulette

smart contracts
preliminary audit report
for internal use only

December 2022

hashex.org

contact@hashex.org

# Contents

# 1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn’t say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

# 2. Overview

HashEx was commissioned by the BNBPot team to perform an audit of their smart contracts. The audit was conducted between 2022-12-22 and 2022-12-26.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at 0x9E3Bc7BE65363A76660c0CBeb77B7A7cD72c264A and 0x8636543727979726302F4E8A4f9922BA1BD7A1C0ba in the Binance Smart Chain (BSC).

Update. The BNBPot team has responded to this report, the updated code is located at 0xBEb72897E1f18D8d9B0C0B868d164B7275D981a3 and 0x7F6eDd97ff44770A71C5Ab127994Ff435233c0f6 in BSC.

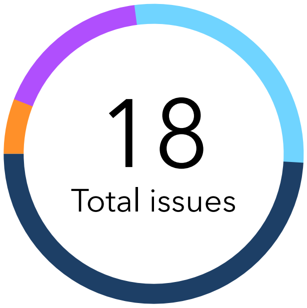
## 2.1 Summary

| Project name | BNBPot Roulette |
|---|---|
| URL | https://bnbpot.io |
| Platform | Binance Smart Chain |
| Language | Solidity |

## 2.2 Contracts

| Name | Address |
|---|---|
| RouletteCasinoNFT | `0x7F6eDd97ff44770A71C5Ab127994Ff435233c0f6` |
| RoulettePot | `0xBEb72897E1f18D8d9B0C0B868d164B7275D981a3` |

# 3. Found issues



| | | |
|---|---|---|
| ● | High | 1 (6%) |
| ● | Medium | 3 (17%) |
| ● | Low | 5 (28%) |
| ● | Info | 9 (49%) |

## C1. RouletteCasinoNFT

| ID | Severity | Title | Status |
|---|---|---|---|
| C1-01 | ● Low | No checks of input parameters | ✓ Partially fixed |
| C1-02 | ● Info | Fixed recipient for minting | ✓ Acknowledged |

## C2. RoulettePot

| ID | Severity | Title | Status |
|---|---|---|---|
| C2-01 | ● High | Winnings' payments aren't guaranteed | ✓ Resolved |
| C2-02 | ● Medium | Error in calculation of the maximum reward for bets | ✓ Resolved |
| C2-03 | ● Medium | Bets can be placed with unsupported numbers | ✓ Acknowledged |
| C2-04 | ● Medium | Swaps with 100% slippage and no deadline | ✓ Acknowledged |
| C2-05 | ● Low | No indexed params in events | ✓ Acknowledged |

| | | | |
|---|---|---|---|
| C2-06 | ● Low | Result of token transfers is not checked | Partially fixed |
| C2-07 | ● Low | Lack of events | Resolved |
| C2-08 | ● Low | Gas optimizations | Partially fixed |
| C2-09 | ● Info | Tokens with taxable transfers aren't supported | Acknowledged |
| C2-10 | ● Info | VRFConsumer source of randomness is out of the scope | Acknowledged |
| C2-11 | ● Info | Typos | Open |
| C2-12 | ● Info | Constructor lacks validation of input parameters | Partially fixed |
| C2-13 | ● Info | Irrelevant comment | Resolved |
| C2-14 | ● Info | Determined random in unused function | Resolved |
| C2-15 | ● Info | Price calculations differs for BNB and ERC20 | Resolved |
| C2-16 | ● Info | No specified visibility | Resolved |

# 4. Contracts

## C1. RouletteCasinoNFT

### Overview

An ERC-721 standard token containing information about the supported payment token. Minting is allowed only for the contract owner, the `mint()` function reports the newly minted tokenId to the RoulettePot contract.

### Issues

#### C1-01 No checks of input parameters ● Low ✓ Partially fixed

The function `mint()` used to mint casino's NFTs does not check the sanity of the input parameters.

```
function mint(
     string memory tokenURI,
     address newTokenAddress,
     string calldata tokenName,
     uint256 maxBet,
     uint256 minBet,
     uint256 fee
 ) external onlyOwner {
...
}
```

The casino's fee can be set to 100% and more.

### Recommendation

Add an upper limit to the fee value. If 100% fees are desirable, explicitly document it in the project's code and whitepater. Check if the `maxBet` is bigger than `minBet`.

### Update

The updated contract allows 99% fees.

### C1-02 Fixed recipient for minting

● Info ✓ Acknowledged

The `mint()` function can be called only by the contract owner, minting recipient is fixed to an `msg.sender` address. NFTs transfers are allowed but the gas cost is higher in the case of `mint(owner) + transfer(recipient)` scenario being compared to `mint(recipient)`.

```
function mint(...) external onlyOwner {
  ...
  _mint(msg.sender, newItemId);
  ...
}
```

# C2. RoulettePot

## Overview

Roulette simulator with the Chainlink VRF as a source of randomness. Parameters for the bets are stored in the RouletteCasinoNFT tokens.

## Issues

### C2-01 Winnings' payments aren't guaranteed

● High ✓ Resolved

`initializeTokenBet()` and `initializeEthBet()` compare the current liquidity to new bets: `require(maxReward <= casinoInfo.liquidity + totalBetAmount)`, but since bets can't be finished immediately (fulfilling randomness takes time), there is a possibility of concurrent userprofitable bets that may deplete the liquidity. The expected value of casino-profit is positive, but there's no guarantee that an arbitrary user will be able to receive his winning.

There is also a possibility to withdraw liquidity with the function `removeLiquidty()` any time for the casino owner. In such a case users won't get their rewards.

### Recommendation

The maximum possible loss should be accounted for in the current liquidity calculation. The `removeLiquidty()` function must respect these temporarily locked tokens.

## C2-02 Error in calculation of the maximum reward for bets ● Medium ✓ Resolved

The function `getMaximumReward()` is used to calculate the maximum reward of a user for given bets. It iterates over all possible numbers and checks which outcome the user will get the biggest reward with. But it does not calculate bets for the number 37.

```
function getMaximumReward(Bet[] memory bets) public pure returns (uint256) {
    uint256 maxReward;
    uint8[6] memory betRewards = [2, 3, 3, 2, 2, 36];

    for (uint256 i = 0; i < 37; i++) {
        for (uint256 j = 0; j < bets.length; j++) {
            if (_isInBet(bets[j], i)) {
                reward += bets[j].amount * betRewards[bets[j].betType];
            }
        }
        if (maxReward < reward) {
            maxReward = reward;
        }
    }
    return maxReward;
}
```

However, according to the function `_isInBet()`, the number 37 is considered as a valid bet.

```
    function _isInBet(Bet memory b, uint256 number) public pure returns (bool) {
        require(b.betType <= 5, 'Invalid bet type');
        require(b.number <= 37, 'Invalid betting number');

        if (number == 0 || number == 37) {
            if (b.betType == 5) {
                return b.number == number;
            } else {
                return false;
            }
        }
```

### Recommendation

Fix the calculations in the `getMaximumReward()` function.

## C2-03 Bets can be placed with unsupported numbers

● Medium ✓ Acknowledged

The `initializeTokenBet()` and `initializeEthBet()` functions don't check the input bets for correctness, i.e. bet number for a given bet type. The `_isInBet()` function simply ignores the numbers greater than 1 or 2 for any type of bet expect the `betType == 5`. Uniformed users may lose their tokens.

### Recommendation

Filter the input data with requirements with meaningful error codes and add the documentation (increase the NatSpec descriptions) for anyone who wants to interact with the contract directly. We also recommend using an enum for the bet type to eliminate the possibility of an inevitably lost bet with an unsupported type.

## C2-04 Swaps with 100% slippage and no deadline

● Medium ✓ Acknowledged

Swaps are performed without `amountMin` and `deadline` parameters, making them vulnerable to sandwich transactions.

## Recommendation

Consider implementing the slippage and deadline in the function parameters.

## Update

Slippage and deadline can't be calculated on-chain, the only option is to receive it from user in function parameters.

## C2-05 No indexed params in events ● Low ✓ Acknowledged

There are no indexed params in the events. Indexed params help to filter events and ease

working with them off-chain.

```
    event RouletteSpinned(
        uint256 tokenId,
        uint256 betId,
        address player,
        uint256 nonce,
        uint256 totalAmount,
        uint256 rewardAmount,
        uint256 totalUSD,
        uint256 rewardUSD,
        uint256 maximumReward
);
    event TransferFailed(uint256 tokenId, address to, uint256 amount); //@audit indexed
params
    event TokenSwapFailed(uint256 tokenId, uint256 balance, string reason, uint256
timestamp);
```

## Recommendation

We recommend adding indexed params to token id, bet id, player parameters in the events.

## C2-06 Result of token transfers is not checked

● Low ✓ Partially fixed

The result of token transfers is not checked in the code. The ERC20 standard states that a transfer function of the token must return a boolean value that shows if the token transfer is successful. Although most implementations of the ERC20 tokens fail on non-successful transfers, there could be implementations that do not fail and return false.

### Recommendation

Use OpenZeppelin's library SafeERC20 to handle token transfers.

## C2-07 Lack of events

● Low ✓ Resolved

Functions `setMaxBet()`, `setMinBet()`, `initializeTokenBet()`, `initializeEthBet()`, `addLiquidtyWithTokens()` and `addLiquidtyWithEth()` don't emit events, complicating the tracking of important off-chain changes.

## C2-08 Gas optimizations

● Low ✓ Partially fixed

1. `uint240` could be used as an amount in the Bet structure to reduce its size to 1 slot.
2. Multiple reads from the storage of the `casinoCount` variable in the `getCasinoList()` and `swapProfitFees()` functions.
3. Multiple reads from the storage of the `casinoInfo.liquidity` variable in the `removeLiquidty()` function.
4. Unchecked math could be used in the `getCasinoList()` and `removeLiquidty()` functions to save gas.
5. Redundant/unreachable code in `_getRandomNumber()` function, `require(amount == msg.value)` in the `addLiquidtyWithEth()`, `require(casinoInfo.liquidity >= amount)` in `removeLiquidty()`, `require(BNB_BUSD_Pair != address(0)` in `getTokenUsdPrice()`, `BNBP_pair = factory.getPair(router.WETH(), BNBPAddress)` in `swapProfitFees()`.
6. No need in a `router.WETH()` call in the `getTokenUsdPrice()` function since `wbnbAddr` is stored and used in `BNB_BUSD_Pair` calculation.

7. `BNB_BUSD_Pair`, used in the `getTokenUsdPrice()` function should be stored as constant or immutable.
8. `require(BNB_Token_Pair != address(0))` in the `getTokenUsdPrice()` function should be checked upon minting of NFT token.
9. `require(BNBP_pair != address(0))` in the `swapProfitFees()` should be checked in the constructor section.
10. `pair = IPancakePair(factory.getPair(router.WETH(), busdAddr))` in the `getBNBPrice()` function should use `wbnbAddr` instead of `router.WETH()` and be stored as constant.
11. `path[1] = router.WETH()` in the `swapProfitFees()` function should be either replaced with `wbnbAddr`, or `wbnbAddr` should be initialized as immutable to `router.WETH()` during the construction.
12. A single `type(uint256).max` approve could be used for swapProfitFees() transfers.
13. Multiple reads from storage of `casinoCount` variable in the `getCasinoList()` and `swapProfitFees()` functions.
14. The internal function `_getRandomNumber()` is not used in the code.

### Update

3 separate calls of `approve()` remain in the `swapProfitFees()` function.

## C2-09 Tokens with taxable transfers aren't supported

● Info ✓ Acknowledged

The contract doesn't support tokens with any type of transfer fees, i.e. the actual transferred amounts aren't checked.

## C2-10 VRFConsumer source of randomness is out of the scope

● Info ✓ Acknowledged

The source of randomness is in the contract at the `vrfConsumer` address; it is out of the scope of this audit. A compromised random may lead to rigged roulette results.

## C2-11 Typos ● Info ? Open

Typos reduce the code's readability. Typos in 'Liquidty', 'retreive', 'Spinned'.

## C2-12 Constructor lacks validation of input parameters ● Info ✓ Partially fixed

The contract constructor does not check the input parameters against zeroes.

## C2-13 Irrelevant comment ● Info ✓ Resolved

NatSpec description of the `_spinWheel()` function states that it generates a random number, but the code denies it.

## C2-14 Determined random in unused function ● Info ✓ Resolved

The `_getRandomNumber()` function uses the determined data as a random one, but this function is not in use. We recommend removing the unused code even if it's not included in the deployed bytecode.

## C2-15 Price calculations differs for BNB and ERC20 ● Info ✓ Resolved

`getTokenUsdPrice()` and `getBNBPrice()` quote an ERC20 token and BNB in BUSD tokens, but the calculations are different: `getAmountsOut()` is used for an ERC20-to-BUSD price and pair reserves fraction is used for a BNB-to-BUSD price. We recommend sticking with a single chosen method even if the calculation error is negligible.

## C2-16 No specified visibility ● Info ✓ Resolved

Variables without specified visibility: `totalBetSum`, `wbnbAddr`, `busdAddr`, `pancakeFactoryAddr`, `pancakeRouterAddr`, `coordinatorAddr`, `linkTokenAddr`, `pegSwapAddr`, `link677TokenAddr`, and `subscriptionId`. Default value `internal` is used. We recommend always explicitly specifying visibility even if it matches with default.

# 5. Conclusion

1 high, 3 medium, 5 low severity issues were found during the audit. 1 high, 1 medium, 1 low issues were resolved in the update.

The reviewed contracts are highly dependent on the owner’s account. Users using the project have to trust the owner and that the owner's account is properly secured.

This audit includes recommendations on code improvement and the prevention of potential attacks.

# Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow.  Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

# Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

# HashEx

BLOCKCHAIN SECURITY