Linea SpinGame V2 Audit Report

Prepared by Cyfrin

Version 2.1

Lead Auditors

Immeas

Hans

June 30, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Linea SpinGame is part of the Linea ecosystem, designed to provide additional incentives to users in the form of ERC20, ERC721, or native tokens. It is a spin/lottery-style contract that utilizes Gelato VRF to generate randomness. All user interactions require off-chain signatures from a trusted service before being validated on-chain. Additionally, the protocol team has the ability to boost the win probability for certain participants, increasing their chances of receiving a prize.

Changes made in this audit include the removal of the consolation NFT and making the contracts upgradeable.

4.1 Actors and Roles

1. Actors:

- **Protocol Team:** Manages the contracts and provides prizes. Ensures that the contract has a sufficient balance of ERC20, ERC721, or native tokens to pay out prizes.
- **Off-Chain Service:** Signs user requests for participation and prize claims.
- **Users:** Participate in the Linea ecosystem and have a chance to win prizes.

2. Roles:

- `CONTROLLER`:
  - Can update prizes.
- `DEFAULT_ADMIN_ROLE`:
  - Can update core contract settings, including:
    - Changing the Gelato VRF Operator
    - Updating the signer address
    - Withdrawing ERC20, ERC721, and native tokens from the contract

        - Cancelling expired participation requests (pending for more than 1 hour)
    - Can assign new controllers.
- `signer`:
    - A wallet responsible for signing user interactions and applying boost values.

4.2 Key Components

- **SpinGame:** The core contract responsible for handling the lottery and prize distribution.

4.3 Centralization Risks

The `CONTROLLER` and `DEFAULT_ADMIN_ROLE` accounts have extensive privileges, including full access to the contract and the ability to withdraw funds.

Additionally, the `signer` wallet has the ability to spam participation requests, which could deplete available prizes and drain the protocol's Gelato balance.

The contract is also upgradeable, giving the admin the power to modify logic post-deployment. While useful for fixes and upgrades, this introduces trust assumptions around the upgrade path and admin governance.

To mitigate risks, all privileged wallets must be securely managed to prevent unauthorized access or potential takeovers.

4.4 ERC20 Token Concerns

The protocol team noted that some ERC20 prize tokens may include common DeFi protocol tokens. Some DeFi tokens are rebasing and can decrease in balance over time, leading to potential depreciation in value.

If rebasing tokens are used as prizes, the team should monitor the contract's balance to ensure it remains sufficient for prize payouts.

5 Audit Scope

contracts/src/Spin.sol

6 Executive Summary

Over the course of 4 days, the Cyfrin team conducted an audit on the Linea SpinGame V2 smart contracts provided by Linea. In this period, a total of 11 issues were found.

During the audit, two low-severity issues were identified. One could cause underflow of `lotAmount` if prizes are misconfigured, and the other relates to global state usage for asynchronous VRF requests, which may result in users receiving unintended prizes or boosts.

Several informational findings were reported, including misleading comments, upgradeability concerns, overcomplicated logic in `_addPrizes`, and minor state cleanup issues. Gas optimizations were also suggested, such as caching repeated storage reads to reduce unnecessary gas costs.

The test suite was thorough and well-structured, covering both success paths and revert paths.

After the audit concluded, the team removed the prize-claim signing mechanism. This update was reviewed in PR #578 (commits `e81485a` and `1365a4c`). Subsequent commits, `5704000` and `4466d17`, also cleaned up the remaining state in `cancelParticipation` and `_fulfillRandomness`. Both changes were audited and deemed safe.

4466d17 was the final merged commit. The compiled bytecode of Spin.sol, using the foundry.toml configuration, was verified to match the bytecode committed in SpinGame-2025-06-30.json.

Summary

Project Name	Linea SpinGame V2
Repository	linea-hub
Commit	0af327319636...
Fix Commit	4466d17de780...
Audit Timeline	Jun 24th - Jun 28th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	6
Gas Optimizations	3
Total Issues	11

Summary of Findings

[L-1] Lack of validation when updating prizes can lead to `lotAmount` underflow when randomness is fulfilled	Resolved
[L-2] Asynchronous VRF request fulfillment uses stale or incorrect parameters due to lack of request-specific data tracking	Acknowledged
[I-1] Overcomplicated `_addPrizes` function designed for incremental updates but only used for complete prize resets	Resolved
[I-2] Incorrect and misleading comments throughout the codebase create confusion	Resolved
[I-3] Stale request id mapping persists for losing spins	Resolved
[I-4] Missing `_disableInitializers()` in constructor	Resolved
[I-5] `GelatoVRFConsumerBase` is not upgrade-safe	Resolved
[I-6] `latestPrizeId` name is misleading	Resolved
[G-1] Cache `signer` in `SpinGame::participate`	Resolved
[G-2] Cache `prize.probability` before first use	Resolved

[G-3] Cache `prize.amount` in `SpinGame::_transferPrize`	Resolved

7 Findings

7.1 Low Risk

7.1.1 Lack of validation when updating prizes can lead to `lotAmount` underflow when randomness is fulfilled

Description: In `SpinGame::_fulfillRandomness`, there's an `unchecked` block that decrements `prize.lotAmount` without validating its current value:

```
/// Should never underflow due to earlier check.
unchecked {
    prize.lotAmount -= 1;
}

if (prize.lotAmount == 0) {
    totalProbabilities -= prizeProbability;
    prize.probability = 0;
}
```

However, the comment claiming that the underflow is prevented by "an earlier check" is outdated as the check was removed in commit `db6ae3d`, making the assumption incorrect. As a result, if a prize is added with a `lotAmount` of 0, the unchecked decrement will underflow to `type(uint32).max`.

Impact: This underflow results in the prize appearing to have an extremely large `lotAmount`, allowing users to repeatedly win and claim that prize well beyond the intended quantity. While this requires a misconfiguration by an admin (e.g. setting `lotAmount = 0`), such errors are plausible in practice and could go unnoticed.

Proof of Concept: The following test demonstrates the underflow in `lotAmount` when a prize is registered with `lotAmount = 0`:

```
function testFulfillRandomnessWith0LotAmount() external {
    MockERC20 token = new MockERC20("Test Token", "TST");
    ISpinGame.Prize[] memory prizesToUpdate = new ISpinGame.Prize[](1);
    uint256[] memory empty = new uint256[](0);

    prizesToUpdate[0] = ISpinGame.Prize({
        tokenAddress: address(token),
        amount: 500 * 1e18,
        lotAmount: 0,
        probability: 1e8,
        availableERC721Ids: empty
    });

    vm.prank(controller);
    spinGame.updatePrizes(prizesToUpdate);

    uint64 nonce = 1;
    uint64 expirationTimestamp = uint64(block.timestamp + 1);
    uint64 boost = 1e8;

    ISpinGame.ParticipationRequest memory request = ISpinGame.ParticipationRequest({
        user: user,
        expirationTimestamp: expirationTimestamp,
        nonce: nonce,
        boost: boost
    });

    bytes32 messageHash = spinGame.hashParticipationExt(request);
    (uint8 v, bytes32 r, bytes32 s) = vm.sign(signer, messageHash);
    ISpinGame.Signature memory signature = ISpinGame.Signature(r, s, v);

    assertFalse(spinGame.nonces(user, nonce));
```

```
    vm.prank(user);
    uint256 requestId = spinGame.participate(nonce, expirationTimestamp, boost, signature);
    assertTrue(spinGame.nonces(user, nonce));

    bytes memory extraData = new bytes(0);
    bytes memory data = abi.encode(requestId, extraData);

    uint256 round = 15608646;
    bytes memory dataWithRound = abi.encode(round, data);

    vm.prank(vrfOperator);
    spinGame.fulfillRandomness(2, dataWithRound);

    uint32[] memory prizeIds = new uint32[](1);
    prizeIds[0] = 0;
    uint256[] memory amounts = spinGame.getUserPrizesWon(user, prizeIds);
    assertEq(amounts[0], 1);
    assertEq(spinGame.hasWonPrize(user, 0), true);

    ISpinGame.Prize memory prize = spinGame.getPrize(0);
    assertEq(prize.lotAmount,type(uint32).max);
}
```

Recommended Mitigation: To prevent this underflow, validate that `lotAmount > 0` when registering new prizes in `_addPrizes`:

```
    uint32 lotAmount = _prizes[i].lotAmount;

+   if (lotAmount == 0) {
+       revert InvalidLotAmount();
+   }

    if (prizeAmount == 0) {
        if (erc721IdsLen != lotAmount) {
            revert MismatchERC721PrizeAmount(erc721IdsLen, lotAmount);
        }
    } else {
        if (erc721IdsLen != 0) {
            revert ERC20PrizeWrongParam();
        }
    }
```

This ensures that any prize expected to be distributed has a non-zero quantity, eliminating the possibility of underflow during fulfillment.

Linea: Fixed in commit `02d6d57`

Cyfrin: Verified. `lotAmount` no verified to be non-zero when a new prize is added.

7.1.2 Asynchronous VRF request fulfillment uses stale or incorrect parameters due to lack of request-specific data tracking

Description: The `SpinGame` contract has a critical architectural flaw in how it handles asynchronous VRF requests. The contract stores user boost values and prize configurations globally rather than tying them to specific requests, leading to inconsistent game behavior when prize structures change or users make multiple participation requests.

There are two primary issues:

First, prize configurations lack versioning. The `SpinGame::updatePrizes` function completely resets the prize structure by deleting existing `prizeIds` and resetting `totalProbabilities` to zero before adding new prizes. When a user calls `SpinGame::participate`, they receive a signature based on the current prize structure. However, if `SpinGame::updatePrizes` is called between the request and VRF fulfillment, the `SpinGame::_fulfillRandomness`

function will use the updated prize structure instead of the one the user expected when participating. This means users could win entirely different prizes than what was available when they participated.

Second, request-specific data is not properly tracked. The `SpinGame::participate` function stores the user's boost value in `userToBoost[user]`, but this mapping gets overwritten if the same user participates again with a different boost before the first request is fulfilled. When `SpinGame::_fulfillRandomness` executes, it retrieves the boost using `uint64 userBoost = userToBoost[user]`, which may not be the boost value from the original request. This creates scenarios where a user who participated with a 150% boost could have their request fulfilled with a 500% boost if they made a second participation with higher boost before the first VRF callback.

The contract only tracks minimal request data through `requestIdToUser` and `requestIdTimestamp` mappings, but fails to capture the complete context needed for proper request fulfillment including the specific boost value and prize structure version at request time.

Impact: Users may receive different prizes or win probabilities than expected when they participated, leading to unfair game outcomes and potential loss of funds.

```
// Scenario 1: Prize structure changes between request and fulfillment
1. User calls participate() when Prize A (50% chance) and Prize B (30% chance) are available
2. Controller calls updatePrizes() with Prize C (60% chance) and Prize D (20% chance)
3. VRF fulfills the request using new prize structure with C and D instead of A and B

// Scenario 2: Multiple participations with different boosts
1. User calls participate() with 150% boost, gets requestId1
2. User calls participate() with 500% boost, gets requestId2
3. VRF fulfills requestId1 but uses 500% boost instead of 150%
```

Recommended Mitigation: Implement request-specific data tracking with prize versioning:

```
+ uint256 public prizeVersion;
+ mapping(uint256 requestId => uint64 boost) public requestIdToBoost;
+ mapping(uint256 requestId => uint256 prizeVersion) public requestIdToPrizeVersion;

function updatePrizes(Prize[] calldata _prizes) external onlyController {
    delete prizeIds;
    totalProbabilities = 0;
+   prizeVersion++;
    _addPrizes(_prizes);
}

function participate(
    uint64 _nonce,
    uint256 _expirationTimestamp,
    uint64 _boost,
    Signature calldata _signature
) external returns (uint256) {
    // ... existing validation ...

-   userToBoost[user] = _boost;
    uint256 requestId = _requestRandomness("");

    requestIdToUser[requestId] = user;
    requestIdTimestamp[requestId] = block.timestamp;
+   requestIdToBoost[requestId] = _boost;
+   requestIdToPrizeVersion[requestId] = prizeVersion;

    // ... rest of function ...
}

function _fulfillRandomness(
    uint256 _randomness,
    uint256 _requestId,
    bytes memory
```

```
) internal override {
    address user = requestIdToUser[_requestId];
    if (user == address(0)) {
        revert InvalidRequestId(_requestId);
    }

+   // Verify prize version hasn't changed
+   if (requestIdToPrizeVersion[_requestId] != prizeVersion) {
+       revert PrizeVersionMismatch();
+   }

-   uint64 userBoost = userToBoost[user];
+   uint64 userBoost = requestIdToBoost[_requestId];

    // ... rest of fulfillment logic ...

+   delete requestIdToBoost[_requestId];
+   delete requestIdToPrizeVersion[_requestId];
}
```

Linea: Acknowledged. We have discussed L2 issue internally. We decided to not fix it because the prizes will be generally the same just with new allocations. In the case of different prizes, we are okay with the behavior that users can win different prizes.

7.2 Informational

7.2.1 Overcomplicated `_addPrizes` function designed for incremental updates but only used for complete prize resets

Description: The `SpinGame::_addPrizes` function was originally designed to incrementally add new prizes to an existing prize pool while preserving existing prizes. However, the function is only called by `SpinGame::updatePrizes`, which first completely resets the prize state by calling `delete prizeIds` and setting `totalProbabilities = 0`. This makes the preservation logic in `_addPrizes` unnecessary and creates several inefficiencies:

1. The function copies the existing `prizeIds` array into `existingArray`, which is always empty after the reset
2. It creates a new array with size `len + existingArrayLen` where `existingArrayLen` is always 0
3. It loops through the empty existing array to copy non-existent prize IDs

The current implementation effectively replaces the entire prize pool but retains incremental addition logic, left over from a removed `SpinGame::addBatchPrizes` function, which no longer serves a meaningful purpose.

Impact: The overcomplicated implementation increases gas costs and code complexity without providing any functional benefit since the incremental addition logic is never utilized.

Recommended Mitigation: Since `updatePrizes` always performs a complete reset, we can simplify `_addPrizes` to remove preservation logic and rename it to `_setPrizes`.

Linea: Fixed in PR#558, commits `0d8611d`, `f9b1bd2`, `e115331`, `2d619e1`, `7920d00`

Cyfrin: Verified.

- `updatePrizes` renamed to `setPrizes`
- `existingArray` logic removed, no iterating over the previous array.
- delete of `prizeIds` removed, `prizeIds` instead overwritten with the memory array `newPrizeIds`

7.2.2 Incorrect and misleading comments throughout the codebase create confusion

Description: The SpinGame contract and ISpin interface contain multiple incorrect, misleading, and inconsistent comments that do not accurately describe the functionality of the code they document. These comments create confusion for developers, auditors, and future maintainers of the codebase.

1. **Line 458 in** `SpinGame::_operator()`: The comment states "Returns the address of the dedicated _msgSender()" which is incorrect. The function returns the `vrfOperator` address, not anything related to `_msgSender()`. This function specifies which address is authorized to call the VRF fulfillment callback. We guess this comment was copied from the `GelatoVRFConsumerBase` without careful consideration.
2. **Line 123 in** `SpinGame::getPrize()`: The `@param _prizeId` comment states "Id of the prize to claim" which is misleading since this is a view function that only returns prize information, not a claiming function.
3. **Line 132 in** `SpinGame::getUserPrizesWon()`: The `@return` comment states "Amounts of prize won" when the function actually returns the count/number of times each prize was won, not token amounts.
4. **Line 59**: The mapping comment states "(userAddress => prizeId => amountOfPrizeIdWon)" but uses "amount" terminology when it actually tracks the number of times a prize was won.
5. **Line 32**: The comment "Last used prizeId" is misleading because `latestPrizeId` represents the next available prize ID, not the last used one.
6. **Line 157 in** `SpinGame::getPrizesAmount()`: The `@return` comment states "PrizesAmount The amount of prizes available" but the variable name suggests it's a count, not an amount. This creates ambiguity about whether it refers to quantity or value.
7. **Lines 290, 300, 309**: The admin withdraw functions are labeled as `@dev Controller function` but they actually require `DEFAULT_ADMIN_ROLE`, not `CONTROLLER_ROLE`.

8. **Line 465**: The `@param _prizes` comment states "Prizes to update" which doesn't clearly indicate this function completely replaces all existing prizes rather than updating them.

Impact: Incorrect documentation can lead to implementation errors, security vulnerabilities, and wasted development time as developers may rely on misleading comments instead of analyzing the actual code behavior.

Recommended Mitigation: Update all misleading comments to accurately reflect the actual functionality:

```
- /// @notice Returns the address of the dedicated _msgSender().
+ /// @notice Returns the address of the VRF operator authorized to fulfill randomness requests.

- /// @param _prizeId Id of the prize to claim.
+ /// @param _prizeId Id of the prize to retrieve information for.

- /// @return Amounts of prize won.
+ /// @return Number of times each prize was won by the user.

- /// @notice Last used prizeId.
+ /// @notice Next available prize ID to be assigned.

- /// @dev Controller function to withdraw ERC20 tokens from the contract.
+ /// @dev Admin function to withdraw ERC20 tokens from the contract.

- /// @param _prizes Prizes to update.
+ /// @param _prizes Prizes to replace all existing prizes with.
```

Linea: Fixed in commits `9f9d9fd` and `b407331`.

Cyfrin: Verified. Comments corrected.

7.2.3 Stale request id mapping persists for losing spins

Description: One change from the previous version of `SpinGame` is the removal of the "LuckyNFT," which served as a consolation prize for users who did not win a "proper" prize.

As part of this change, the clearing of the request mapping (`requestIdToUser`) was moved so that it now occurs **only** in the prize win path, as shown in `SpinGame::_fulfillRandomness`:

```
            userToPrizesWon[user][selectedPrizeId] += 1;
            delete requestIdToUser[_requestId];
            emit PrizeWon(user, selectedPrizeId);

            return;
        }
    }
    emit NoPrizeWon(user);
}
```

However, it is considered good housekeeping to clear the `requestIdToUser` mapping regardless of whether the user wins or not. This ensures consistent state cleanup and prevents potential stale entries from persisting.

Consider unconditionally clearing the request mapping as follows:

```
function _fulfillRandomness(
    uint256 _randomness,
    uint256 _requestId,
    bytes memory
) internal override {
    address user = requestIdToUser[_requestId];
    if (user == address(0)) {
        revert InvalidRequestId(_requestId);
    }
+   delete requestIdToUser[_requestId];
```

```
    ...

            userToPrizesWon[user][selectedPrizeId] += 1;
-           delete requestIdToUser[_requestId];
            emit PrizeWon(user, selectedPrizeId);


            return;
        }
    }
    emit NoPrizeWon(user);
}
```

Linea: Fixed in commit 9f9d9fd

Cyfrin: Verified. `requestIdToUser` deleted at the beginning of `_fulfillRandomness`.

7.2.4 Missing `_disableInitializers()` in constructor

Description: The `SpinGame` contract is upgradeable but does **not** call `_disableInitializers()` in its constructor. In upgradeable contract patterns, this call is a best practice to prevent the implementation (logic) contract from being initialized directly.

While this doesn't affect the proxy's behavior, it helps protect against accidental or malicious use of the implementation contract in isolation, especially in environments where both proxy and implementation contracts are visible, like block explorers.

Consider adding the following line to the constructor of the `SpinGame` contract:

```
constructor(address _trustedForwarderAddress) ERC2771ContextUpgradeable(_trustedForwarderAddress) {
    _disableInitializers();
}
```

This ensures that the implementation contract cannot be initialized independently.

Linea: Fixed in commit 9f9d9fd

Cyfrin: Verified. Constructor now calls `_disableInitializers`.

7.2.5 `GelatoVRFConsumerBase` is not upgrade-safe

Description: The `SpinGame` contract has been changed to be upgradeable. But, the contract inherits from `GelatoVRFConsumerBase`, which is not upgrade-safe as it lacks a reserved storage gap (`uint256[x] private __gap`) to prevent future storage collisions. This poses a risk if the `GelatoVRFConsumerBase` contract is modified upstream (e.g. adds state variables), it could lead to storage layout corruption during upgrades.

Since `GelatoVRFConsumerBase` is an external dependency outside the control of the audited codebase, this risk cannot be addressed directly within that contract.

However to mitigate the risk, consider adding a storage buffer to account for potential future changes in `GelatoVRFConsumerBase`. This can be done by:

1. Adding a `_gap` in `SpinGame` itself:

   ```
   uint256[x] private __gelatoBuffer;
   ```

2. Alternatively, inserting an intermediate "buffer" contract in the inheritance chain that exists solely to reserve storage space:

   ```
   contract GelatoVRFGap {
       uint256[x] private __gap;
   }
   ```

```
contract SpinGame is ..., GelatoVRFConsumerBase, GelatoVRFGap
```

This ensures that even if `GelatoVRFConsumerBase` adds storage in the future, it won't overwrite critical storage slots in `SpinGame`.

Linea: Fixed in commit `9f9d9fd`

Cyfrin: Verified, first storage slots in the contract now is `uint256[50] private __gelatoBuffer`

7.2.6 `latestPrizeId` name is misleading

Description: As part of changes in this audit, the `latestPrizeId` variable now functions as a counter for the next available prize ID, rather than tracking the most recently used one as in the previous version. This creates a naming mismatch, since `latestPrizeId` no longer reflects its actual behavior.

The function `_addPrizes` even caches it as `nextPrizeId`, which is a more accurate description. Consider renaming it as `nextPrizeId`.

Linea: Fixed in commits `8266a91` and `dd8de7b`

Cyfrin: Verified. Field renamed to `nextPrizeId` and stack variable in `__addPrizes` renamed to `currentPrizeId`.

7.3 Gas Optimization

7.3.1 Cache `signer` in `SpinGame::participate`

Description: In `SpinGame::participate`, the `signer` state variable is accessed multiple times:

```
address recoveredSigner = ECDSA.recover(...);
if (recoveredSigner != signer || signer == address(0)) {
    revert SignerNotAllowed(recoveredSigner);
}
```

In contrast, `SpinGame::claimPrize` caches `signer` to a local variable (`cachedSigner`) before comparison. This avoids redundant storage reads, which are more expensive than local memory accesses.

Considerer caching `signer` in a local variable at the start of the relevant check in `SpinGame::participate` as well.

Linea: Fixed in commit 0290123

Cyfrin: Verified. `signer` is cached and the cached value is used in the comparisons.

7.3.2 Cache `prize.probability` before first use

Description: In `SpinGame::_fulfillRandomness()`, `prize.probability` is first read in an `if` statement, and then cached immediately afterward:

```
if (prize.probability == 0) {
    continue;
}
uint64 prizeProbability = prize.probability;
```

This results in an unnecessary initial storage read before caching. Consider caching it above the `if`-statement.

Linea: Fixed in commit 0290123

Cyfrin: Verified. The caching of `prize.probability` is moved above the `if` and the cached value is used in the comparison.

7.3.3 Cache `prize.amount` in `SpinGame::_transferPrize`

Description: In `SpinGame::_transferPrize`, when transferring either an ERC20 or native token prize, the `prize.amount` field is read three times: once for the `if` check, once to compare against the contract balance, and again when executing the transfer:

```
if (prize.amount > 0) {
    ...
    if (contractBalance < prize.amount) {
        revert PrizeAmountExceedsBalance(..., prize.amount, contractBalance);
    }
    ...
    token.safeTransfer(_winner, prize.amount);
}
```

This results in three separate reads of the same storage slot. Since the value does not change during execution, it can be cached once.

Consider caching `prize.amount` at the head of the first if:

```
uint256 amount = prize.amount;
if (amount > 0) {
    ...
    if (contractBalance < amount) {
        revert PrizeAmountExceedsBalance(..., amount, contractBalance);
    }
```

```
    ...
    token.safeTransfer(_winner, amount);
}
```

Linea: Fixed in commit 0290123

Cyfrin: Verified. `prize.amount` is now cached and the cached value is used.