Dec 4, 2024

ZAN

Security Assessment
Volare Battle

—

Professional Service

Table of Contents

1. Overview

1.1. Executive Summary

Volare Battle is a project that rewards participants for predicting bull or bear markets. This report has been prepared for Volare Battle to discover issues and vulnerabilities in the source code of this project as well as any contract dependencies that were not part of an officially recognized library.

Conducted by Static Analysis, Formal Verification and Manual Review, we have identified **1 Low and 5 Informational issues** in commit 989dbb3.

The project team has resolved **all issues except for issue described in L-01** in commit ab45328. The project team acknowledged the issue described in L-01 and has confirmed that this implementation aligns with the business design. Therefore, they decided to keep no change.

1.2. Project Summary

Project Name	Volare Battle
Platform	Ethereum
Language	Solidity
Codebase	Audit 1: • https://github.com/VolareFinance/volare-battle/tree/989dbb323411949ebc5b8250e4b32cc5ec5fa38a Final Audit: • https://github.com/VolareFinance/volare-battle/tree/ab45328901b460c9f94a75413e2bfd766a7ce7a9

1.3. Assessment Summary

Delivery Date	Dec 4, 2024
Audit Methodology	Static Analysis, Formal Verification, Manual Review

1.4. Assessment Scope

ID	File	File Hash
1	/src/Battle.sol	7bb13e349b8ccd7728df4d707e2a81

2. Checklist

2.1. Code Security

Reentrancy	DelegateCall	Integer Overflow
Input Validation	Unchecked this.call	Frozen Money
Arbitrary External Call	Unchecked Owner Transfer	Do-while Continue
Right-To-Left-Override Character	Unauthenticated Storage Access	Risk For Weak Randomness
TxOrigin	Missing Checks for Return Values	Diamond Inheritance
ThisBalance	VarType Deduction	Array Length Manipulation
Uninitialized Variable	Shadow Variable	Divide Before Multiply
Affected by Compiler Bug		

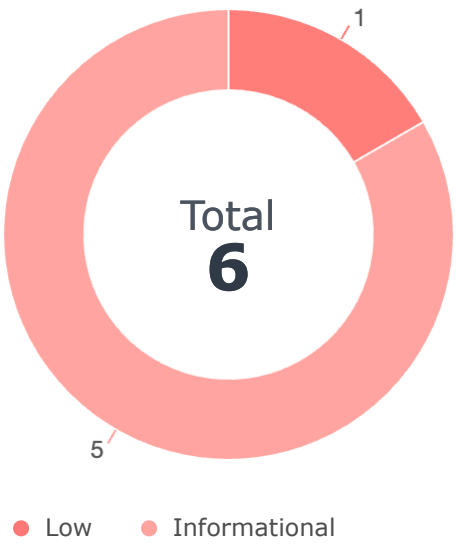
2.2. Optimization Suggestion

Compiler Version	Improper State Variable Modification
Function Visibility	Deprecated Function
Externally Controlled Variables	Code Style
Constant Specific	Event Specific
Return Value Unspecified	Inexistent Error Message
State Variable Defined Without Storage Location	Import Issue
Compare With Timestamp/Block Number/Blockhash	Constructor in Base Contract Not Implemented
Delete Struct Containing the Mapping Type	Usage of '=+'
Paths in the Modifier Not End with "_" or Revert	Non-payable Public Functions Use msg.value
Lack of SafeMath	Compiler Error/Warning
Tautology Issue	Loop Depends on Array Length
Redundant/Duplicated/Dead Code	Code Complexity/Code Inefficiency
Undeclared Resource	Optimizable Return Statement
Unused Resource	

2.3. Business Security

The Code Implementation is Consistent With Comments, Project White Papers and Other Materials
Permission Check
Address Check

3. Findings



Code Security

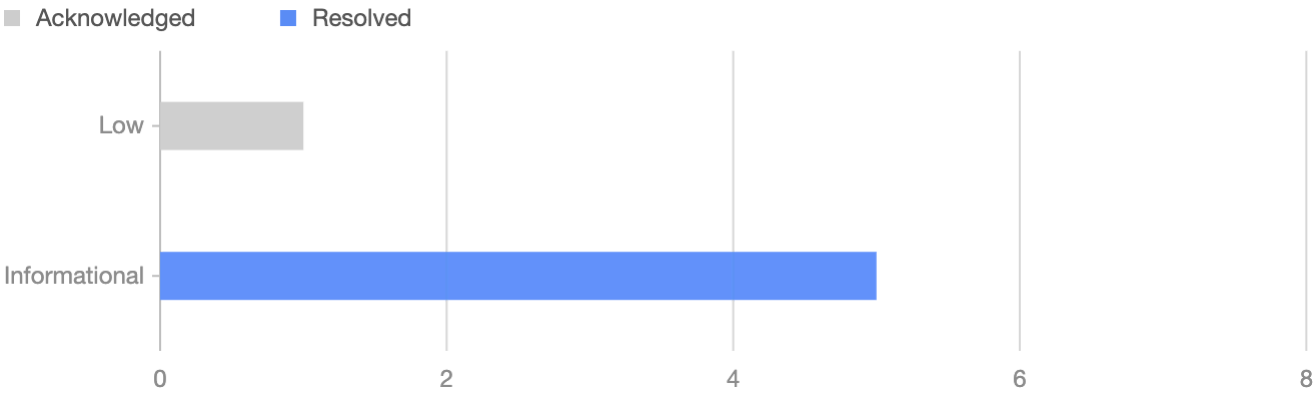
no vulnerabilities.

Optimization Suggestion

Informational: 5

Business Security

Low: 1



ID	Title	Category	Severity	Status
L-01	Some Tokens Are Frozen in the Contract	Business Security	Low	Acknowledged
I-02	No Check of Address Params with Zero Address	Optimization Suggestion	Informational	Resolved
I-03	Use storage Instead of memory for Struct or Array	Optimization Suggestion	Informational	Resolved
I-04	Set the Constant to Private	Optimization Suggestion	Informational	Resolved
I-05	Incorrect Comment	Optimization Suggestion	Informational	Resolved
I-06	Duplicate Check in Contract	Optimization Suggestion	Informational	Resolved

L-01: Some Tokens Are Frozen in the Contract

Low: Business Security

File Location: /src/Battle.sol:222

Description

In the `claim` function, users can claim rewards. When calculating the amount of rewards that the user can claim, `addedReward`, there is a precision loss in the calculation at line 222 due to Solidity's integer division rounding down. This can result in users being unable to withdraw all their rewards (`round.rewardAmount`). However, the remaining rewards that could not be claimed cannot be extracted through any other function in the `Battle` contract, and this portion of the rewards will be frozen in the `Battle` contract.

POC

1. Assuming the token's decimal is 6, in a certain epoch, user A and user B both called the `bear` function, each wagering 1.112112 tokens, while User C called the `bull` function, wagering 1.112114 tokens.
2. In this epoch, the `closePrice` was less than the `lockPrice`, so users A and B won.
3. Calculate the rewards for this epoch. Assuming the `treasuryFee` is 1000, the calculated `treasuryAmt` will be 0.333633 tokens, the `rewardBaseCalAmount` is 2.224224 tokens, and the `rewardAmount` is 3.002705 tokens.
4. User A withdraws the reward. The withdrawable reward amount is calculated as `addedReward = 1112112 * 3002705 / 2224224 = 1501352`, which means user A can withdraw 1.501352 tokens. Similarly, user B can also withdraw 1.501352 tokens.
5. The winning users A and B collectively withdrew 3.002704 tokens, while 0.000001 token (3.002705 - 3.002704) is frozen in the `Battle` contract and cannot be withdrawn.

/src/Battle.sol

```
211   function claim(uint256[] calldata epochs) external nonReentrant
      notContract {
212       uint256 reward; // Initializes reward
213
214       for (uint256 i = 0; i < epochs.length; i++) {
215           require(rounds[epochs[i]].startTimestamp != 0, "Round has not
              started");
216           require(block.timestamp > rounds[epochs[i]].closeTimestamp,
              "Round has not ended");
217           uint256 addedReward = 0;
218           // Round valid, claim rewards
219           if (rounds[epochs[i]].oracleCalled) {
220               require(claimable(epochs[i], msg.sender), "Not eligible for
                  claim");
221               Round memory round = rounds[epochs[i]];
222               addedReward = (ledger[epochs[i]][msg.sender].amount * round.
                  rewardAmount) / round.rewardBaseCalAmount;
223           } // Round invalid, refund battle amount
224
```

Recommendation

It is recommended to provide a function to extract tokens that have been left in the contract due to precision loss.

Alleviation

The project team acknowledged the issue and decided to keep no change.

I-02: No Check of Address Params with Zero Address

Informational: Optimization Suggestion

File Location: /src/Battle.sol:132,412

Description

The input parameter of the address type in the function does not use the zero address for verification.

/src/Battle.sol

```
129         * @param _treasuryFee: treasury fee (1000 = 10%)
130         */
131        constructor(
132            IERC20 _token,
133            address _oracleAddress,
```

/src/Battle.sol

```
410         * @dev Callable by owner
411         */
412        function recoverToken(address _token, uint256 _amount) external
           onlyOwner {
413            require(_token != address(token), "Cannot be prediction token
               address");
414            IERC20(_token).safeTransfer(address(msg.sender), _amount);
```

Recommendation

It is recommended to perform zero address verification on the input parameters of the address type.

Alleviation

Resolved in commit ab45328.

I-03: Use storage Instead of memory for Struct or Array

Informational: Optimization Suggestion

File Location: /src/Battle.sol:221,476,477,492,493

Description

When reading a state variable of data type struct or array, declaring a storage variable to receive the struct or array is more gas-efficient than declaring a memory variable. This is because assigning data in storage to a memory variable would cause all fields or elements of the struct or array to be read from storage, even if not all fields or elements are used.

Additionally, reading fields from the new memory variable would incur additional mload operations, thereby increasing gas consumption. For structs, declaring a storage variable to receive the state variable can save approximately 3000 gas. For an array with a length of 1000, declaring a storage variable to receive the state variable can save approximately 2 million gas.

/src/Battle.sol

```
211   function claim(uint256[] calldata epochs) external nonReentrant
      notContract {
212       uint256 reward; // Initializes reward
213
214       for (uint256 i = 0; i < epochs.length; i++) {
215           require(rounds[epochs[i]].startTimestamp != 0, "Round has not
              started");
216           require(block.timestamp > rounds[epochs[i]].closeTimestamp,
              "Round has not ended");
217           uint256 addedReward = 0;
218           // Round valid, claim rewards
219           if (rounds[epochs[i]].oracleCalled) {
220               require(claimable(epochs[i], msg.sender), "Not eligible for
                  claim");
221               Round memory round = rounds[epochs[i]];
222               addedReward = (ledger[epochs[i]][msg.sender].amount * round.
                  rewardAmount) / round.rewardBaseCalAmount;
223           } // Round invalid, refund battle amount
224           else {
225               require(refundable(epochs[i], msg.sender), "Not eligible for
                  refund");
226               addedReward = ledger[epochs[i]][msg.sender].amount;
227           }
228
229           ledger[epochs[i]][msg.sender].claimed = true;
230           reward += addedReward;
231
232           emit Claim(msg.sender, epochs[i], addedReward);
233       }
234
235       if (reward > 0) {
```

```
236             token.safeTransfer(msg.sender, reward);
237         }
238     }
```

/src/Battle.sol

```
474         */
475     function claimable(uint256 epoch, address user) public view returns
        (bool) {
476         BattleInfo memory battleInfo = ledger[epoch][user];
477         Round memory round = rounds[epoch];
478         return round.oracleCalled && battleInfo.amount != 0 && !battleInfo.
            claimed
```

/src/Battle.sol

```
490         */
491     function refundable(uint256 epoch, address user) public view returns
        (bool) {
492         BattleInfo memory battleInfo = ledger[epoch][user];
493         Round memory round = rounds[epoch];
494         return !round.oracleCalled && !battleInfo.claimed && block.timestamp
            > round.closeTimestamp + bufferSeconds
```

Recommendation

It is recommended to declare a storage variable instead of a memory variable when reading a state variable of data type struct or array.

In particular, regarding the aforementioned issue in the `claim` function, we recommend executing `Round storage round = rounds[epochs[i]]` before line 215, and then using `round` in place of `rounds[epochs[i]]` in the subsequent code.

Alleviation

Resolved in commit ab45328.

I-04: Set the Constant to Private

Informational: Optimization Suggestion

File Location: /src/Battle.sol:40

Description

For constants, if the visibility is set to public, the compiler will automatically generate a getter function for it, which will consume more gas during deployment.

/src/Battle.sol

```
38      uint256 public oracleUpdateAllowance; // seconds
39
40      uint256 public constant MAX_TREASURY_FEE = 1000; // 10%
41
42      mapping(uint256 => mapping(address => BattleInfo)) public ledger;
```

Recommendation

It is recommended to set the visibility of constants to private instead of public.

Alleviation

Resolved in commit ab45328.

I-05: Incorrect Comment

Informational: Optimization Suggestion

File Location: /src/Battle.sol:616,617

Description

According to the code logic of the `_canBattle` function and the business logic, the comments on lines 616 and 617 are incorrect. The correct comments should be "Round must have started" and "Current timestamp must be within startTimestamp and lockTimestamp," respectively.

/src/Battle.sol

```
614  /**
615      * @notice Determine if a round is valid for battle
616      * Round must have started and locked
617      * Current timestamp must be within startTimestamp and closeTimestamp
618      */
619  function _canBattle(uint256 epoch) internal view returns (bool) {
620      return rounds[epoch].startTimestamp != 0 && rounds[epoch].
         lockTimestamp != 0
621          && block.timestamp > rounds[epoch].startTimestamp && block.
             timestamp < rounds[epoch].lockTimestamp;
622  }
```

Recommendation

It is recommended to modify the comments on line 616 and 617 to "Round must have started" and "Current timestamp must be within startTimestamp and lockTimestamp," respectively.

Alleviation

Resolved in commit ab45328.

I-06: Duplicate Check in Contract

Informational: Optimization Suggestion

File Location: Battle.sol:246,589

Description

There is duplicate code in the function `executeRound` and `_safeStartRound` which will increase gas consumption. In the `executeRound` function, there are some redundant codes that can be safely removed. The `require(genesisStartOnce && genesisLockOnce, "Can only run after genesisStartRound and genesisLockRound is triggered")` is called (line 245 - line 248) to ensure the `genesisStartOnce` variable is true. However, there is the same protection logic in the `_safeStartRound` function(line 589), and only called by function `executeRound`.

Battle.sol

```
244   function executeRound() external whenNotPaused onlyKeeper {
245           require(
246               genesisStartOnce && genesisLockOnce,
247               "Can only run after genesisStartRound and genesisLockRound
                  is triggered"
248           );
249
250           (uint80 currentRoundId, int256 currentPrice) =
              _getPriceFromOracle();
251
252           oracleLatestRoundId = uint256(currentRoundId);
253
254           // CurrentEpoch refers to previous round (n-1)
255           _safeLockRound(currentEpoch, currentRoundId, currentPrice);
256           _safeEndRound(currentEpoch - 1, currentRoundId, currentPrice);
257           _calculateRewards(currentEpoch - 1);
258
259           // Increment currentEpoch to current round (n)
260           currentEpoch = currentEpoch + 1;
261           _safeStartRound(currentEpoch);
262       }
```

Battle.sol

```
588   function _safeStartRound(uint256 epoch) internal {
589           require(genesisStartOnce, "Can only run after genesisStartRound
              is triggered");
590           require(rounds[epoch - 2].closeTimestamp != 0, "Can only start
              round after round n-2 has ended");
591           require(
592               block.timestamp >= rounds[epoch - 2].closeTimestamp,
593               "Can only start new round after round n-2 closeTimestamp"
594           );
595           _startRound(epoch);
596       }
```

Recommendation

It is recommended to remove duplication of code in contracts to save gas.

Alleviation

Resolved in commit ab45328.

4. Disclaimer

No description, statement, recommendation or conclusion in this report shall be construed as endorsement, affirmation or confirmation of the project. The security assessment is limited to the scope of work as stipulated in the Statement of Work.

This report is prepared in response to source code, and based on the attacks and vulnerabilities in the source code that already existed or occurred before the date of this report, excluding any new attacks or vulnerabilities that exist or occur after the date of this report. The security assessment are solely based on the documents and materials provided by the customer, and the customer represents and warrants documents and materials are true, accurate and complete.

CONSULTANT DOES NOT MAKE AND HEREBY DISCLAIMS ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, REGARDING THE SERVICES, DELIVERABLES, OR ANY OTHER MATTER PERTAINING TO THIS REPORT.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR AND HEREBY DISCLAIMS MERCHANTABILITY, FITNESS FOR PURPOSE, TITLE, NON-INFRINGEMENT OR NON-APPROPRIATION OF INTELLECTUAL PROPERTY RIGHTS OF A THIRD PARTY, SATISFACTORY QUALITY, ACCURACY, QUALITY, COMPLETENESS, TIMELINESS, RESPONSIVENESS, OR PRODUCTIVITY OF THE SERVICES OR DELIVERABLES.

CONSULTANT EXCLUDES ANY WARRANTY THAT THE SERVICES AND DELIVERABLES WILL BE UNINTERRUPTED, ERROR FREE, FREE OF SECURITY DEFECTS OR HARMFUL COMPONENTS, REVEAL ALL SECURITY VULNERABILITIES, OR THAT ANY DATA WILL NOT BE LOST OR CORRUPTED.

CONSULTANT SHALL NOT BE RESPONSIBLE FOR (A) ANY REPRESENTATIONS MADE BY ANY PERSON REGARDING THE SUFFICIENCY OR SUITABILITY OF SERVICES AND DELIVERABLES IN ANY ACTUAL APPLICATION, OR (B) WHETHER ANY SUCH USE WOULD VIOLATE OR INFRINGE THE APPLICABLE LAWS, OR (C) REVIEWING THE CUSTOMER MATERIALS FOR ACCURACY.

5. Appendix

5.1 Visibility

Contract	FuncName	Visibility	Mutability	Modifiers
Battle	_CTOR_	public	Y	
Battle	bear	external	Y	whenNotPaused, nonReentrant, notContract
Battle	bull	external	Y	whenNotPaused, nonReentrant, notContract
Battle	claim	external	Y	nonReentrant, notContract
Battle	executeRound	external	Y	whenNotPaused, onlyKeeper
Battle	genesisLockRound	external	Y	whenNotPaused, onlyKeeper
Battle	genesisStartRound	external	Y	whenNotPaused, onlyKeeper
Battle	pause	external	Y	whenNotPaused, onlyAdminOrKeeper
Battle	claimTreasury	external	Y	nonReentrant, onlyAdmin
Battle	unpause	external	Y	whenPaused, onlyAdminOrKeeper
Battle	setBufferAndIntervalSeconds	external	Y	whenPaused, onlyAdmin
Battle	setMinBattleAmount	external	Y	whenPaused, onlyAdmin
Battle	setKeeper	external	Y	onlyAdmin
Battle	setOracle	external	Y	whenPaused, onlyAdmin
Battle	setOracleUpdateAllowance	external	Y	whenPaused, onlyAdmin

Contract	FuncName	Visibility	Mutability	Modifiers
Battle	setTreasuryFee	external	Y	whenPaused, onlyAdmin
Battle	recoverToken	external	N	onlyOwner
Battle	setAdmin	external	Y	onlyOwner
Battle	getUserRounds	external	N	
Battle	getUserRoundsLength	external	N	
Battle	claimable	public	N	
Battle	refundable	public	N	
Battle	_calculateRewards	internal	N	
Battle	_safeEndRound	internal	N	
Battle	_safeLockRound	internal	N	
Battle	_safeStartRound	internal	N	
Battle	_startRound	internal	N	
Battle	_canBattle	internal	N	
Battle	_getPriceFromOracle	internal	N	
Battle	_isContract	internal	N	

5. Appendix

5.2 Call Graph

Battle

5. Appendix

5.3 Inheritance Graph

Battle

5.4 Formal Verification Metadata

1. Upon the successful execution of the function `bear`, it is imperative that the epoch provided as a parameter is equivalent to the currentEpoch.

```
/// #if_succeeds {:msg "Upon the successful execution of the function
`bear`, it is imperative that the epoch provided as a parameter is
equivalent to the currentEpoch."} epoch == old(currentEpoch);
/**
  * @notice Bear position
  * @param epoch: epoch
  */
function bear(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

2. Should the function `bear` be executed without fail, the round corresponding to the provided epoch must have started and be within the valid battle timeframe.

```
/// #if_succeeds {:msg "Should the function `bear` be executed without
fail, the round corresponding to the provided epoch must have started and
be within the valid battle timeframe."} _canBattle(epoch);
/**
  * @notice Bear position
  * @param epoch: epoch
  */
function bear(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

3. In the event of a successful execution of the function `bear`, the amount transferred by the user must be greater than or equal to the minimum battle amount set for the contract.

```
/// #if_succeeds {:msg "In the event of a successful execution of the
function `bear`, the amount transferred by the user must be greater than or
equal to the minimum battle amount set for the contract."} _amount >=
old(minBattleAmount);
/**
  * @notice Bear position
  * @param epoch: epoch
  */
function bear(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

4. Upon the flawless execution of the function `bear`, the user should not have previously participated in the current round with a non-zero amount.

```
/// #if_succeeds {:msg "Upon the flawless execution of the function `bear`,
the user should not have previously participated in the current round with
a non-zero amount."} old(ledger[epoch][msg.sender].amount) == 0;
/**
  * @notice Bear position
  * @param epoch: epoch
  */
function bear(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

5. Should the function `bear` execute successfully, the total amount and bear amount in the round should be incremented by the amount sent by the user.

```
/// #if_succeeds {:msg "Should the function `bear` execute successfully,
the total amount and bear amount in the round should be incremented by the
amount sent by the user."} rounds[epoch].totalAmount ==
old(rounds[epoch].totalAmount) + _amount && rounds[epoch].bearAmount ==
old(rounds[epoch].bearAmount) + _amount;
/**
  * @notice Bear position
  * @param epoch: epoch
  */
function bear(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

6. Upon the successful execution of the function `bear`, the user's position for the given epoch should be set to Bear, and their battle amount recorded correctly.

```
/// #if_succeeds {:msg "Upon the successful execution of the function
`bear`, the user's position for the given epoch should be set to Bear, and
their battle amount recorded correctly."} ledger[epoch]
[msg.sender].position == Position.Bear && ledger[epoch][msg.sender].amount
== _amount;
/**
  * @notice Bear position
  * @param epoch: epoch
  */
function bear(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

7. In the case where the function `bear` executes without error, the userRounds mapping for the sender should include the current epoch.

```
/// #if_succeeds {:msg "In the case where the function `bear` executes
without error, the userRounds mapping for the sender should include the
current epoch."} userRounds[msg.sender].length ==
old(userRounds[msg.sender].length) + 1 && userRounds[msg.sender]
[userRounds[msg.sender].length - 1] == epoch;
/**
  * @notice Bear position
  * @param epoch: epoch
  */
function bear(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

8. Should the function `bull` be executed with success, it is imperative that the epoch parameter corresponds to the currentEpoch.

```
/// #if_succeeds {:msg "Should the function `bull` be executed with
success, it is imperative that the epoch parameter corresponds to the
currentEpoch."} epoch == old(currentEpoch);
/**
  * @notice Bull position
  * @param epoch: epoch
  */
function bull(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

9. Upon the successful execution of the function `bull`, it must be verified that the battle for the given epoch is still permissible according to the _canBattle function.

```
/// #if_succeeds {:msg "Upon the successful execution of the function
`bull`, it must be verified that the battle for the given epoch is still
permissible according to the _canBattle function."} _canBattle(epoch);
/**
  * @notice Bull position
  * @param epoch: epoch
  */
function bull(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

10. In the event that the function `bull` executes without fail, the amount provided as a parameter should be equal to or greater than the minBattleAmount.

```
/// #if_succeeds {:msg "In the event that the function `bull` executes
without fail, the amount provided as a parameter should be equal to or
greater than the minBattleAmount."} _amount >= minBattleAmount;
/**
  * @notice Bull position
  * @param epoch: epoch
  */
function bull(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

11. Should the function `bull` execute successfully, it is essential that the sender has not already participated in the current round (epoch) with a non-zero amount.

```
/// #if_succeeds {:msg "Should the function `bull` execute successfully, it
is essential that the sender has not already participated in the current
round (epoch) with a non-zero amount."} old(ledger[epoch]
[msg.sender].amount) == 0;
/**
  * @notice Bull position
  * @param epoch: epoch
  */
function bull(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

12. Upon the successful execution of the function `bull`, the ledger should record the correct position (Bull) and the amount bet by the user for the specified epoch.

```
/// #if_succeeds {:msg "Upon the successful execution of the function
`bull`, the ledger should record the correct position (Bull) and the amount
bet by the user for the specified epoch."} ledger[epoch]
[msg.sender].position == Position.Bull && ledger[epoch][msg.sender].amount
== _amount;
/**
  * @notice Bull position
  * @param epoch: epoch
  */
function bull(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

13. In the case where the function `bull` is executed with success, the round's total amount and bull amount should be updated correctly with the new bet amount.

```
/// #if_succeeds {:msg "In the case where the function `bull` is executed
with success, the round's total amount and bull amount should be updated
correctly with the new bet amount."} rounds[epoch].totalAmount ==
old(rounds[epoch].totalAmount) + _amount && rounds[epoch].bullAmount ==
old(rounds[epoch].bullAmount) + _amount;
/**
  * @notice Bull position
  * @param epoch: epoch
  */
function bull(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

14. Should the function `bull` be executed successfully, the userRounds mapping for the sender should include the current epoch.

```
/// #if_succeeds {:msg "Should the function `bull` be executed
successfully, the userRounds mapping for the sender should include the
current epoch."} userRounds[msg.sender].length ==
old(userRounds[msg.sender].length) + 1 && userRounds[msg.sender]
[userRounds[msg.sender].length - 1] == epoch;
/**
  * @notice Bull position
  * @param epoch: epoch
  */
function bull(uint256 epoch, uint256 _amount) external whenNotPaused
nonReentrant notContract {
```

Passed.

15. In the event that the function `claim` executes successfully, for each epoch in the provided epochs array, the round must have started and ended, and the user must be eligible to claim or refund based on the round's state.

```
/// #if_succeeds {:msg "In the event that the function `claim` executes
successfully, for each epoch in the provided epochs array, the round must
have started and ended, and the user must be eligible to claim or refund
based on the round's state."}
///     forall (uint256 i) in 0..epochs.length {
///         rounds[epochs[i]].startTimestamp != 0 && block.timestamp >
rounds[epochs[i]].closeTimestamp &&
///         (
///             (rounds[epochs[i]].oracleCalled && claimable(epochs[i],
msg.sender)) ||
///             (!rounds[epochs[i]].oracleCalled && refundable(epochs[i],
msg.sender))
///         )
///     };
/**
```

```
* @notice Claim reward for an array of epochs
* @param epochs: array of epochs
*/
function claim(uint256[] calldata epochs) external nonReentrant notContract
{
```

Passed.

16. Should the function `claim` execute without fail, the total reward amount transferred to the user should match the sum of all added rewards for each epoch in the provided epochs array.

```
/// #if_succeeds {:msg "Should the function `claim` execute without fail,
the total reward amount transferred to the user should match the sum of all
added rewards for each epoch in the provided epochs array."}
///     token.balanceOf(msg.sender) == old(token.balanceOf(msg.sender)) +
reward;
/**
* @notice Claim reward for an array of epochs
* @param epochs: array of epochs
*/
function claim(uint256[] calldata epochs) external nonReentrant notContract
{
```

Passed.

17. Upon the successful execution of the function `claim`, for each epoch in the provided epochs array, the user's claimed status for that epoch should be set to true.

```
/// #if_succeeds {:msg "Upon the successful execution of the function
`claim`, for each epoch in the provided epochs array, the user's claimed
status for that epoch should be set to true."}
///     forall (uint256 i) in 0..epochs.length {
///         ledger[epochs[i]][msg.sender].claimed == true
///     };
/**
* @notice Claim reward for an array of epochs
* @param epochs: array of epochs
*/
function claim(uint256[] calldata epochs) external nonReentrant notContract
{
```

Passed.

18. Upon successful execution, the function `executeRound` should only run after both genesisStartOnce and genesisLockOnce are true.

```
/// #if_succeeds {:msg "Upon successful execution, the function
`executeRound` should only run after both genesisStartOnce and
genesisLockOnce are true."} old(genesisStartOnce) && old(genesisLockOnce);
/**
  * @notice Start the next round n, lock price for round n-1, end round n-2
  * @dev Callable by Keeper
  */
function executeRound() external whenNotPaused onlyKeeper {
```

Passed.

19. Upon successful execution, the function `executeRound` should lock the round for the current epoch (n-1), end the round for the previous epoch (n-2), and start a new round for the next epoch (n).

```
/// #if_succeeds {:msg "Upon successful execution, the function
`executeRound` should lock the round for the current epoch (n-1), end the
round for the previous epoch (n-2), and start a new round for the next
epoch (n)."} rounds[old(currentEpoch)].lockTimestamp != 0 &&
rounds[old(currentEpoch) - 1].closePrice != 0 && rounds[old(currentEpoch) +
1].startTimestamp != 0;
/**
  * @notice Start the next round n, lock price for round n-1, end round n-2
  * @dev Callable by Keeper
  */
function executeRound() external whenNotPaused onlyKeeper {
```

Passed.

20. Upon successful execution, the function `executeRound` should increment the current epoch by 1.

```
/// #if_succeeds {:msg "Upon successful execution, the function
`executeRound` should increment the current epoch by 1."} currentEpoch ==
old(currentEpoch) + 1;
/**
  * @notice Start the next round n, lock price for round n-1, end round n-2
  * @dev Callable by Keeper
  */
function executeRound() external whenNotPaused onlyKeeper {
```

Passed.

21. Upon successful execution, the function `executeRound` should update the oracleLatestRoundId with the latest round ID from the oracle.

```
/// #if_succeeds {:msg "Upon successful execution, the function
`executeRound` should update the oracleLatestRoundId with the latest round
```

```
ID from the oracle."} oracleLatestRoundId > old(oracleLatestRoundId);
/**
  * @notice Start the next round n, lock price for round n-1, end round n-2
  * @dev Callable by Keeper
  */
function executeRound() external whenNotPaused onlyKeeper {
```

Passed.

22. Upon successful execution, the function `executeRound` should calculate rewards for the round (n-1) if not already calculated.

```
/// #if_succeeds {:msg "Upon successful execution, the function
`executeRound` should calculate rewards for the round (n-1) if not already
calculated."} rounds[old(currentEpoch) - 1].rewardBaseCalAmount > 0 &&
rounds[old(currentEpoch) - 1].rewardAmount > 0;
/**
  * @notice Start the next round n, lock price for round n-1, end round n-2
  * @dev Callable by Keeper
  */
function executeRound() external whenNotPaused onlyKeeper {
```

Passed.

23. Upon successful execution, the function `executeRound` should ensure that the close price for the round (n-2) is set and the round is marked as called.

```
/// #if_succeeds {:msg "Upon successful execution, the function
`executeRound` should ensure that the close price for the round (n-2) is
set and the round is marked as called."} rounds[old(currentEpoch) -
2].closePrice != 0 && rounds[old(currentEpoch) - 2].oracleCalled;
/**
  * @notice Start the next round n, lock price for round n-1, end round n-2
  * @dev Callable by Keeper
  */
function executeRound() external whenNotPaused onlyKeeper {
```

Passed.

24. Upon successful execution, the function `executeRound` should ensure that the lock price for the round (n-1) is set and the round is marked as locked.

```
/// #if_succeeds {:msg "Upon successful execution, the function
`executeRound` should ensure that the lock price for the round (n-1) is set
and the round is marked as locked."} rounds[old(currentEpoch)].lockPrice !=
0 && rounds[old(currentEpoch)].lockOracleId != 0;
/**
  * @notice Start the next round n, lock price for round n-1, end round n-2
  * @dev Callable by Keeper
  */
```

```
    function executeRound() external whenNotPaused onlyKeeper {
```

Passed.