BriVault report

Prepared by: Guangyang Zhong

Date: 2025.11.12

Contents

Chapter 1

Audit Details

1.1 Scope

```
src/
|-- briTechToken.sol
|-- briVault.sol
```

1.2 Roles

```
Actors:
owner : Only the owner can set the winner after the event ends.
Users : Users have to send in asset to the contract (deposit + participation fee).
        users should not be able to deposit once the event starts.
        Users should only join events only after they have made deposit.
```

Chapter 2

Executive Summary

2.1 Issues found

Severity	Number of issues found
High	5
Medium	1
Low	2

Chapter 3

Findings

3.1 High

3.1.1 [H-1] Function:_setFinallizedVaultBalance target IERC20(asset()).balance as finalizedVaultAsset, ignoring maybe someone has deposited but not joined event.

Description

In `Function:_setFinallizedVaultBalance ()`, set `finalizedVaultAsset` equal `IERC20(asset()).balanceO`

```
    function _setFinallizedVaultBalance () internal returns (uint256) {

        if (block.timestamp <= eventStartDate) {
            revert eventNotStarted();
        }

@>      return finalizedVaultAsset = IERC20(asset()).balanceOf(address(this));
    }
```

Impact

It will let people who has deposited the asset but not joined event loose their asset.

Proof of Concepts

You can run the test function in `test/briVault.t.sol`, and it will pass.

```
/**
 * @dev Test that finalizedVaultAsset incorrectly includes funds from non-participating
↪  users
 * Scenario:
 *   - user1 deposits and joins the event
 *   - user2 deposits but does not join the event
 * Expected:
 *   - finalizedVaultAsset should include only user1's stake
 * Actual:
 *   - finalizedVaultAsset includes user2's funds (bug)
 */

    function test_incorrect_finalizedVaultAsset() public {
        //user1 has deposited and joined event
        mockToken.approve(address(briVault), 5 ether);
        uint256 user1shares = briVault.deposit(5 ether, user1);

        briVault.joinEvent(10);
        console.log("user1 shares", user1shares);
        vm.stopPrank();

        //user2 has deposited but not joined event
        vm.startPrank(user2);
        mockToken.approve(address(briVault), 5 ether);
        briVault.deposit(5 ether, user2);
        vm.stopPrank();

        //setwinner and caculate finalizedVaultAsset
        vm.startPrank(owner);
        briVault.setCountry(countries);
        vm.warp(eventEndDate + 1);
        briVault.setWinner(2);
        vm.stopPrank();

        //unfortunately, finalizedVaultAsset contains user2's assets
        uint256 correctFinalized = briVault.stakedAsset(user1);
        uint256 actualFinalized = briVault.finalizedVaultAsset();
        assertGt(actualFinalized, correctFinalized, "finalizedVaultAsset incorrectly includes
         ↪  non-participant assets");
    }
```

Recommended mitigation

```
    function _setFinallizedVaultBalance () internal returns (uint256) {

        if (block.timestamp <= eventStartDate) {
            revert eventNotStarted();
        }

-          return finalizedVaultAsset = IERC20(asset()).balanceOf(address(this));
+          return finalizedVaultAsset = Math.mulDiv(IERC20(asset()).balanceOf(address(this)),
 ↪  totalParticipantShares, totalSupply());
    }
```

3.1.2 [H-2] In Function:deposit, stakedAsset[receiver] uses = to update insteadof +=, incorrectly updating the state of stakedAsset[receiver]

Description

In Function:deposit, stakedAsset[receiver] uses = to update insteadof +=:

```
    function deposit(uint256 assets, address receiver) public override returns (uint256) {
        require(receiver != address(0));

        if (block.timestamp >= eventStartDate) {
            revert eventStarted();
        }

        ......
```

```
@>        stakedAsset[receiver] = stakeAsset;

        ......

    }
```

Impact

Assume there is no fee. If userA first deposits 5 ether into the contract, and then deposits another 1 ether, stakedAsset[userA] now equals 1 ether instead of 6 ether.

Proof of Concepts

Run the test funtion below in `test/briVault.t.sol`:

```
    function test_incorrect_stakedAsset_update() public {
        uint256 base = 10000;
        uint256 expected_stakedAsset;
        vm.startPrank(user1);

        //user1 first deposit 5 ehter.
        mockToken.approve(address(briVault), 5 ether);
        briVault.deposit(5 ether, user1);
        expected_stakedAsset = 5 ether - (5 ether * participationFeeBsp) / base;
        assertEq(briVault.stakedAsset(user1), expected_stakedAsset);

        //user1 then deposit another 5 ether.
        mockToken.approve(address(briVault), 5 ether);
        briVault.deposit(5 ether, user1);
        expected_stakedAsset += 5 ether - (5 ether * participationFeeBsp) / base;

        //find that actual stakedAsset less than expected_stakedAsset.
        assertLt(briVault.stakedAsset(user1), expected_stakedAsset);
        console.log("actual_stakedAsset", briVault.stakedAsset(user1));
    }
```

Recommended mitigation

```
    function deposit(uint256 assets, address receiver) public override returns (uint256) {
        require(receiver != address(0));

        if (block.timestamp >= eventStartDate) {
            revert eventStarted();
        }

        ......

-        stakedAsset[receiver] = stakeAsset;
+        stakedAsset[receiver] += stakeAsset;
        ......

    }
```

3.1.3 [H-3] In Funciton:`deposit`, `_mint` to `msg.sender` instead of `receiver`

Description

```
    function deposit(uint256 assets, address receiver) public override returns (uint256) {
        require(receiver != address(0));

        if (block.timestamp >= eventStartDate) {
            revert eventStarted();
        }

        ......

@>        _mint(msg.sender, participantShares);

        ......
    }
```

Impact

Assume that userA wants to deposit asset for userB, but actually userA receives minted shares istead of userB.

Proof of Concepts

Run the test function below in `test/briVault.t.sol`:

```
    function test_incorrect_mint_in_deposit() public {
        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        briVault.deposit(5 ether, user2);
        vm.stopPrank();

        assertEq(briVault.balanceOf(user2), 0);
    }
```

Recommended mitigation

```
    function deposit(uint256 assets, address receiver) public override returns (uint256) {
        require(receiver != address(0));

        if (block.timestamp >= eventStartDate) {
            revert eventStarted();
        }

        ......

-        _mint(msg.sender, participantShares);
+        _mint(receiver, participantShares);

        ......
    }
```

3.1.4 [H-4] `Function:joinEvent` has no logic code to tackle reentracy, leading that user can use the same shares to run several times

Description

In the `Function:joinEvent`

```
    function joinEvent(uint256 countryId) public {
            revert noDeposit();
        }

        // Ensure countryId is a valid index in the `teams` array
        if (countryId >= teams.length) {
            revert invalidCountry();
        }

        if (block.timestamp > eventStartDate) {
            revert eventStarted();
        }


        userToCountry[msg.sender] = teams[countryId];


        uint256 participantShares = balanceOf(msg.sender);
        //There is no checks to reuse of participantShares
@>        userSharesToCountry[msg.sender][countryId] = participantShares;

        usersAddress.push(msg.sender);

        numberOfParticipants++;

        totalParticipantShares += participantShares;

        emit joinedEvent(msg.sender, countryId);
    }
```

Impact

Assume tha there are 48 countries, userA only deposits 5 ethers and get 5 shares, then userA runs `Function:joinEvent` several times, finally userA has all countries' shares, no matter which countries win, he always get awards.

Proof of Concepts

Run the test function below in `test/briVault.t.sol`, you can add more `briVault.joinEvent(any_number);`, and change `briVault.setWinner(any_number);`, user1 will always withdraw assets.

```
    function test_reentrancy_joinEvent() public {
        uint256 expected_assetToWithdraw;
        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        briVault.deposit(5 ether, user1);
        //user1 can join more than 1 country.
        briVault.joinEvent(0);
        briVault.joinEvent(1);
        briVault.joinEvent(2);
        briVault.joinEvent(3);
        briVault.joinEvent(4);
        briVault.joinEvent(5);
        vm.stopPrank();

        vm.startPrank(user2);
        mockToken.approve(address(briVault), 5 ether);
        briVault.deposit(5 ether, user2);
        briVault.joinEvent(0);
        vm.stopPrank();

        vm.warp(eventEndDate + 1);
        vm.startPrank(owner);
        briVault.setWinner(0);
        //briVault.setWinner(1);
        //briVault.setWinner(2);
        //etc, no matter which number is Winner, user1 always get rewards.
        expected_assetToWithdraw = briVault.balanceOf(user1) * briVault.finalizedVaultAsset()
 ↪  / briVault.totalWinnerShares();
        vm.expectEmit(address(briVault));
        emit BriVault.Withdraw(user1, expected_assetToWithdraw);
        vm.startPrank(user1);
        briVault.withdraw();
        vm.stopPrank();

    }
```

Recommended mitigation

1. You can check if user has already joined through `userCountryId`

```
+ uint16 constant UNSET_COUNTRY = type(uint16).max;
+ mapping(address => uint16) public userCountryId; // default UNSET_COUNTRY


    function joinEvent(uint256 countryId) public {
        if (stakedAsset[msg.sender] == 0) {
            revert noDeposit();
        }

        // Ensure countryId is a valid index in the `teams` array
        if (countryId >= teams.length) {
            revert invalidCountry();
        }

        if (block.timestamp > eventStartDate) {
            revert eventStarted();
        }
+       // Check if user has already joined
+       uint16 existing = userCountryId[msg.sender];
+       if (existing != UNSET_COUNTRY) revert alreadyJoined();
+
+       // Record the user's country
+       userCountryId[msg.sender] = uint16(countryId);

        userToCountry[msg.sender] = teams[countryId];

        uint256 participantShares = balanceOf(msg.sender);
        userSharesToCountry[msg.sender][countryId] = participantShares;

        usersAddress.push(msg.sender);

        numberOfParticipants++;

        totalParticipantShares += participantShares;
        emit joinedEvent(msg.sender, countryId);
    }
```

2. Or you can use unusedsharesToUser to prevent user from reusing

3.1.5 [H-5] `Function:cancelParticipation` only `_burn` shares, but the others related to shares are left unchanged

Description

In the `Function:cancelParticipation` user only `_burn` shares:

```
    function cancelParticipation () public  {

        if (block.timestamp >= eventStartDate){
           revert eventStarted();
        }

        uint256 refundAmount = stakedAsset[msg.sender];

        stakedAsset[msg.sender] = 0;

         uint256 shares = balanceOf(msg.sender);

@>         _burn(msg.sender, shares);

        IERC20(asset()).safeTransfer(msg.sender, refundAmount);
    }
```

But if this user has joined event, `Function:joinEvent` update some variables related to his shares:

```
    function joinEvent(uint256 countryId) public {
        ......

        userToCountry[msg.sender] = teams[countryId];


        uint256 participantShares = balanceOf(msg.sender);
@>        userSharesToCountry[msg.sender][countryId] = participantShares;

        usersAddress.push(msg.sender);

        numberOfParticipants++;

@>        totalParticipantShares += participantShares;

        emit joinedEvent(msg.sender, countryId);
    }
```

Impact

Assume that userA deposits some assets and attains some share, then userA join event, if he choose countryId 0, userSharesToCountry[userA][0] = userA's shares, after that userA cancelParticipation, his balance in briVault is burned, but userSharesToCountry[userA][0] isn't set 0, and totalParticipantShares is also not set correctly.

Run the test function below in `test/briVault.t.sol`:

```
    /**
     *Scenario:
        -user1: deposit and joinEvent
        -user2: deposit, joinEvent and cancelParticipation

     *Expected:
        -user1 withdraw IERC20(address(mockToken)).balanceOf(address(briVault)) back

     *Actual:
        -user1 only withdraw half of IERC20(address(mockToken)).balanceOf(address(briVault))
     */
    function test_cancelParticipation_incorrect_shares_update() public {
        uint256 expected_assetToWithdraw;
        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        briVault.deposit(5 ether, user1);
        briVault.joinEvent(0);
        vm.stopPrank();

        vm.startPrank(user2);
        mockToken.approve(address(briVault), 5 ether);
        briVault.deposit(5 ether, user2);
        briVault.joinEvent(0);
        briVault.cancelParticipation();
        vm.stopPrank();

        vm.warp(eventEndDate + 1);
        vm.startPrank(owner);
        briVault.setWinner(0);
        expected_assetToWithdraw = IERC20(address(mockToken)).balanceOf(address(briVault));
        vm.expectEmit(address(briVault));
        emit BriVault.Withdraw(user1, expected_assetToWithdraw);
        vm.startPrank(user1);
        briVault.withdraw();
        vm.stopPrank();
    }
```

Recommended mitigation

```
    function cancelParticipation () public  {

        if (block.timestamp >= eventStartDate){
           revert eventStarted();
        }
        //q: maybe need some checks?
        uint256 refundAmount = stakedAsset[msg.sender];

        stakedAsset[msg.sender] = 0;

        uint256 shares = balanceOf(msg.sender);
+        totalParticipantShares -= shares;
+        for (uint256 i = 0; i < teams.length; ++i) {
+            userSharesToCountry[msg.sender][i] = 0;
+        }
```

```
        _burn(msg.sender, shares);


        IERC20(asset()).safeTransfer(msg.sender, refundAmount);
    }
```

3.2 Medium

3.2.1 [M-2] Users who deposited but did not join the event cannot withdraw their assets

Description

The `withdraw()` function currently allows only participants who selected the **winner country** to withdraw rewards. However, users who **deposited assets but never joined the event** (i.e., `userToCountry[msg.sender` is unset) have no way to retrieve their deposited tokens once the event ends. These users remain permanently locked out of their funds.

Impact

- **Loss of funds** for users who accidentally deposited but didn't call the join function.
- Causes user dissatisfaction and potential trust issues.
- Contradicts expected UX behavior — deposits should be refundable if participation was never completed.

Proof of Concepts

```
function withdraw() external winnerSet {
    if (block.timestamp < eventEndDate) {
        revert eventNotEnded();
    }

    // Users who never joined have userToCountry[msg.sender] == ""
    // This will revert in didNotWin() and permanently lock their funds.
    if (
        keccak256(abi.encodePacked(userToCountry[msg.sender])) !=
        keccak256(abi.encodePacked(winner))
    ) {
        revert didNotWin();
    }

    uint256 shares = balanceOf(msg.sender);
    ...
}
```

Recommended mitigation

Add a branch allowing users who deposited but never joined to withdraw their assets after event ends. For example:

```
function withdraw() external winnerSet {
    if (block.timestamp < eventEndDate) {
        revert eventNotEnded();
    }
    string memory joinedCountry = userToCountry[msg.sender];

    // Allow refund if user never joined
    if (bytes(joinedCountry).length == 0) {
        uint256 refund = stakedAsset[msg.sender];
        stakedAsset[msg.sender] = 0;
        IERC20(asset()).safeTransfer(msg.sender, refund);
        emit Refund(msg.sender, refund);
        return;
    }

    // Otherwise only winners can withdraw reward
    if (
        keccak256(abi.encodePacked(joinedCountry)) !=
        keccak256(abi.encodePacked(winner))
    ) {
        revert didNotWin();
    }

    uint256 shares = balanceOf(msg.sender);
    uint256 assetToWithdraw =
        Math.mulDiv(shares, finalizedVaultAsset, totalWinnerShares);

    _burn(msg.sender, shares);
    IERC20(asset()).safeTransfer(msg.sender, assetToWithdraw);
    emit Withdraw(msg.sender, assetToWithdraw);
}
```

This ensures:

- Depositors who never joined can still get refunds.
- Participants follow the winner/loser withdrawal logic.

3.3 Low

3.3.1 [L-1] `Function:deposit` lack asset check, leading meaningless function operation

Description

The deposit function calculates the participation fee and attempts to transfer assets from the user, but it does not explicitly check whether the user has enough token balance before proceeding:

```
uint256 fee = _getParticipationFee(assets);
// No check for IERC20(asset()).balanceOf(msg.sender) >= assets
```

Impact

1. If the user has insufficient balance, safeTransferFrom will revert.
2. The function may appear to succeed in the logic flow before revert, leading to misleading expectations.
3. Lacks clear, early feedback to the user that their balance is insufficient.

```
+uint256 balance = IERC20(asset()).balanceOf(msg.sender);
+if (balance < assets) {
+    revert("Insufficient balance to deposit");
+}
```

3.3.2 [L-2] `Function:cancelParticipation` lacks checks for stakedAsset

Description

The cancelParticipation() function allows users to withdraw their staked assets before the event starts. However, the function does not perform sufficient checks before proceeding:

```
uint256 refundAmount = stakedAsset[msg.sender];
//lacks checks for stakedAsset
stakedAsset[msg.sender] = 0;

uint256 shares = balanceOf(msg.sender);
_burn(msg.sender, shares);

IERC20(asset()).safeTransfer(msg.sender, refundAmount);
```

Impact

A user with no deposit could still attempt to call the function, wasting gas or triggering revert unexpectedly.

Recommended mitigation

```
+if (stakedAsset[msg.sender] == 0) revert noDeposit();
```

Chapter 4

Informational

4.1 Function:_setFinalizedVaultBalance () time check conflicts with Function:setWinner() thime check

Description In Function:_setFinalizedVaultBalance (), check logic is

```
if (block.timestamp <= eventStartDate) {
    revert eventNotStarted();
}
```

But in Function:setWinner(), check logic is

```
    function setWinner(uint256 countryIndex) public onlyOwner returns (string memory) {


        ......


@>        if (block.timestamp <= eventEndDate) {
            revert eventNotEnded();
        }


@>        _setFinalizedVaultBalance();


        ......
    }
```

It is obvious that eventEndDate > eventStartDate.

Recommended mitigation

```
    function _setFinalizedVaultBalance () internal returns (uint256) {
-        if (block.timestamp <= eventStartDate) {
+        if (block.timestamp <= eventEndDate) {
            revert eventNotStarted();
        }
        return finalizedVaultAsset = IERC20(asset()).balanceOf(address(this));
    }
```

4.2 L-1: Centralization Risk

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform

5 Found Instances

- Found in src/briTechToken.sol Line: 7

```
contract BriTechToken is ERC20, Ownable {
```

- Found in src/briTechToken.sol Line: 10

```
    function mint() public onlyOwner {
```

- Found in src/briVault.sol Line: 13

```
contract BriVault is ERC4626, Ownable {
```

- Found in src/briVault.sol Line: 107

```
 function setCountry(string[48] memory countries) public onlyOwner {
```

- Found in src/briVault.sol Line: 117

```
    function setWinner(uint256 countryIndex) public onlyOwner returns (string memory) {
```

4.3 L-2: Unspecific Solidity Pragma

Consider using a specific version of Solidity in your contracts instead of a wide version. For example, instead of `pragma solidity ^0.8.0;`, use `pragma solidity 0.8.0;`

2 Found Instances

- Found in src/briTechToken.sol Line: 2

```
pragma solidity ^0.8.24;
```

- Found in src/briVault.sol Line: 3

```
pragma solidity ^0.8.24;
```

4.4 L-3: Address State Variable Set Without Checks

Check for `address(0)` when assigning values to address state variables.

1 Found Instances

- Found in src/briVault.sol Line: 89

```
        participationFeeAddress = _participationFeeAddress;
```

4.5 L-4: Public Function Not Used Internally

If a function is marked public but is not used internally, consider marking it as `external`.

6 Found Instances

```
    function mint() public onlyOwner {
```

- Found in src/briVault.sol Line: 107

```
function setCountry(string[48] memory countries) public onlyOwner {
```

- Found in src/briVault.sol Line: 117

```
    function setWinner(uint256 countryIndex) public onlyOwner returns (string memory) {
```

- Found in src/briVault.sol Line: 174

```
    function getWinner () public view returns (string memory) {
```

- Found in src/briVault.sol Line: 244

```
    function joinEvent(uint256 countryId) public {
```

- Found in src/briVault.sol Line: 277

```
    function cancelParticipation () public  {
```

4.6 L-5: Empty `require()` / `revert()` Statement

Use descriptive reason strings or custom errors for revert paths.

1 Found Instances

- Found in src/briVault.sol Line: 210

```
        require(receiver != address(0));
```

4.7 L-6: PUSH0 Opcode

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

2 Found Instances

- Found in src/briTechToken.sol Line: 2

```
pragma solidity ^0.8.24;
```

- Found in src/briVault.sol Line: 3

```
pragma solidity ^0.8.24;
```

4.8 L-7: Modifier Invoked Only Once

Consider removing the modifier or inlining the logic into the calling function.

1 Found Instances

- Found in src/briVault.sol Line: 94

```
    modifier winnerSet () {
```

4.9 L-8: Large Numeric Literal

Large literal values multiples of 10000 can be replaced with scientific notation.Use `e` notation, for example: `1e18`, instead of its full numeric value.

2 Found Instances

- Found in src/briTechToken.sol Line: 11

```
        _mint(owner(), 10_000_000 * 1e18);
```

- Found in src/briVault.sol Line: 19

```
    uint256 constant BASE = 10000;
```

4.10 L-9: Unused Error

Consider using or removing the unused error.

1 Found Instances

- Found in src/briVault.sol Line: 63

```
    error notRegistered();
```

4.11 L-10: Local Variable Shadows State Variable

Rename the local variable that shadows another state variable.

1 Found Instances

- Found in src/briVault.sol Line: 81

```
    constructor (IERC20 _asset, uint256 _participationFeeBsp, uint256 _eventStartDate,
↪   address _participationFeeAddress, uint256 _minimumAmount, uint256 _eventEndDate)
↪   ERC4626 (_asset) ERC20("BriTechLabs", "BTT") Ownable(msg.sender) {
```

4.12 L-11: Storage Array Length not Cached

Calling `.length` on a storage array in a loop condition is expensive. Consider caching the length in a local variable in memory before the loop and reusing it.

1 Found Instances

- Found in src/briVault.sol Line: 195

```
        for (uint256 i = 0; i < usersAddress.length; ++i){
```

4.13 L-12: Costly operations inside loop

Invoking `SSTORE` operations in loops may waste gas. Use a local variable to hold the loop computation result.

2 Found Instances

- Found in src/briVault.sol Line: 108

```
    for (uint256 i = 0; i < countries.length; ++i) {
```

- Found in src/briVault.sol Line: 195

```
        for (uint256 i = 0; i < usersAddress.length; ++i){
```

4.14 L-13: State Variable Could Be Immutable

State variables that are only changed in the constructor should be declared immutable to save gas. Add the `immutable` attribute to state variables that are only changed in the constructor

5 Found Instances

- Found in src/briVault.sol Line: 17

```
    uint256 public participationFeeBsp;
```

- Found in src/briVault.sol Line: 24

```
    address private participationFeeAddress;
```

- Found in src/briVault.sol Line: 26

```
    uint256 public eventStartDate;
```

- Found in src/briVault.sol Line: 28

```
    uint256 public eventEndDate;
```

- Found in src/briVault.sol Line: 48

```
    uint256 public  minimumAmount;
```

4.15 L-14: Unchecked Return

Function returns a value but it is ignored. Consider checking the return value.

2 Found Instances

- Found in src/briVault.sol Line: 134

  ```
          _getWinnerShares();
  ```

- Found in src/briVault.sol Line: 136

  ```
          _setFinallizedVaultBalance();
  ```

Chapter 5

Gas

5.1 [G-1] `_getWinnerShares()` unnecessarily reads `usersAddress.len` from storage on each iteration and may repeatedly accumulate `totalWinnerShares`

Description The internal function `_getWinnerShares()` iterates over `usersAddress` and reads `usersAddress.l` directly from storage in each loop condition check. Accessing storage in a loop significantly increases gas costs. Additionally, since the function uses `+=` to update `totalWinnerShares` without resetting it to zero, repeated calls may cause the total shares to be **counted multiple times**, leading to incorrect reward distribution.

Impact

- Unnecessary gas consumption due to repeated storage reads.
- Risk of overcounting `totalWinnerShares` if `_getWinnerShares()` is invoked more than once, resulting in **inflated reward shares** and incorrect asset withdrawals.

Proof of Concepts

```
function _getWinnerShares () internal returns (uint256) {
    for (uint256 i = 0; i < usersAddress.length; ++i){
        address user = usersAddress[i];
        totalWinnerShares += userSharesToCountry[user][winnerCountryId];
    }
    return totalWinnerShares;
}
```

Each `i < usersAddress.length` reads from storage every iteration. Also, if `_getWinnerShares()` is called multiple times, `totalWinnerShares` will be cumulatively increased again.

Recommended mitigation

1. Cache the array length in a local variable before entering the loop.
2. Use a local accumulator variable and assign it to `totalWinnerShares` after the loop to prevent repeated accumulation.

```
function _getWinnerShares() internal returns (uint256) {
-   for (uint256 i = 0; i < usersAddress.length; ++i){
-       address user = usersAddress[i];
```

```diff
-   }
-   return totalWinnerShares;

+   uint256 len = usersAddress.length; // cache length
+   uint256 total = 0;
+   for (uint256 i = 0; i < len; ++i) {
+       address user = usersAddress[i];
+       total += userSharesToCountry[user][winnerCountryId];
+   }
+   totalWinnerShares = total;
+   return total;
}
```

5.2 [G-2] Redundant modifier logic — should be wrapped into reusable internal check function

Description The `winnerSet` modifier directly implements the conditional check `if (_setWinner != true)` inside its body. This pattern causes the compiler to inline the logic each time the modifier is applied, increasing contract bytecode size and slightly raising deployment gas cost.

Impact While functionally correct, this design leads to:

- Unnecessary bytecode bloat
- Increased deployment gas
- Harder to maintain if similar logic appears in other modifiers or functions

Proof of Concepts

```
modifier winnerSet() {
    if (_setWinner != true) {
        revert winnerNotSet();
    }
    _;
}
```

Each time this modifier is attached to a function, Solidity copies its logic inline.

Recommended mitigation Refactor the condition into an **internal reusable function**, then call it inside the modifier. This will reduce repetitive bytecode and centralize the check logic:

```
function _requireWinnerSet() internal view {
    if (!_setWinner) revert winnerNotSet();
}

modifier winnerSet() {
    _requireWinnerSet();
    _;
}
```

This reduces contract size and simplifies future maintenance when new modifiers or checks are added.