



BriVault Audit Report

Version 1.0

Elia Bordoni

December 1, 2025

BriVault Audit Report

Elia Bordoni

December 1, 2025

Prepared by: Elia Bordoni

Table of Contents

- Table of Contents
- Protocol Summary
- Risk Classification
- Audit Details
 - Scope
 - Roles
- Executive Summary
 - Issues found
- Findings
 - High
 - * [H-01] BriVault joinEvent function allows multiple calls from the same address, potentially causing a DoS due to unbounded growth of the BriVault:usersAddress array
 - * [H-02] Inconsistent Share Ownership When Depositing to a Different Receiver
 - Medium
 - Low
 - Informational
 - Gas

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. 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.

Risk Classification

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

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

Commit hash:

c28c0e1451cdcf3b97ae74260e8af718bdcf2749

Scope

./src/

- briTechToken.sol
- beiVault.sol

Roles

1. Owner:

- RESPONSIBILITIES:
 - Only the owner can set the winner after the event ends.
- LIMITATIONS:
 - Owner cannot participate to the event

2. Users:

- RESPONSIBILITIES:
 - Users have to send in asset to the contract (deposit + participation fee).
 - Users should only join events only after they have made deposit.
- LIMITATIONS:
 - Users should not be able to deposit once the event starts.

Executive Summary

The entire audit was carried out exclusively through manual review.

Issues found

Severity	Number of issues found
High	2
Medium	0
Low	0
Total	2

Findings

High

[H-01] BriVault joinEvent function allows multiple calls from the same address, potentially causing a DoS due to unbounded growth of the BriVault:usersAddress array

Description

- Once a user has called `BriVault:deposit` and has assets staked, they can call `BriVault:joinEvent` to place their bet.
However, after the bet is placed, the user should not be able to call `BriVault:joinEvent` again unless they first cancel their previous participation.
- The issue lies in the fact that `BriVault:joinEvent` does not check whether a user is already registered, allowing multiple registrations with a single deposit. A malicious actor could exploit this by repeatedly calling `BriVault:joinEvent` thousands of times, causing the `BriVault:usersAddress` array to grow excessively large and potentially leading to a Denial of Service (DoS) when iterating over it in a `for` loop.

```
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();
    }
    //does not check if msg.sender is already registered

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

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

```
@gt;    usersAddress.push(msg.sender);

        numberOfParticipants++;
        totalParticipantShares += participantShares;

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

Likelihood:

- Whenever a malicious user wants to break the contract and holds at least `BriVault:minimumAmountToJoin` plus the required fee.

Impact:

- Permanent DoS: When the owner attempts to call `BriVault:setWinner`, the transaction will always revert because the for loop inside the internal function `BriVault:_getWinnerShares` will run out of gas. As a result, the contract will become unusable, and all the underlying tokens will remain permanently locked.

Proof of Concept In the test, an account named `attacker` performs a deposit on `BriVault:deposit` and then calls `BriVault:joinEvent` **35,000 times**, causing the `BriVault:usersAddress` array to grow excessively large. As a result, when the owner later calls `BriVault:setWinner`, the call reverts due to running out of gas while iterating the array inside `_getWinnerShares`, producing a permanent DoS and leaving the underlying tokens locked.

The setup is the same as the tests performed by the developer.

```
function testDOSAttackOnUsersAddressArray() public {
    uint256 depositAmount = 1 ether;
    vm.startPrank(attacker);
    mockToken.approve(address(briVault), depositAmount);
    briVault.deposit(depositAmount, attacker);
    vm.stopPrank();
    for(uint256 i = 0; i < 35000; i++) {
        vm.prank(attacker);
        briVault.joinEvent(0);
    }

    vm.warp(briVault.eventEndDate() +1);

    vm.startPrank(owner);
    uint256 gasLimit = 30_000_000;
    uint256 gasBefore = gasleft();
```

```

        briVault.setWinner(0);
        uint256 gasUsed = gasBefore - gasleft();

        vm.stopPrank();
        assertGt(gasLimit, gasUsed, "OutOfGas");

    }

```

Recommended Mitigation

To mitigate this issue, one possible (but inefficient) approach would be to iterate through the `BriVault:usersAddress` array on each call to verify whether the address has already joined. However, this solution would be too gas-intensive.

A more efficient approach is to introduce a mapping (for example, `mapping(address => bool)`) that tracks whether an address has already joined the event. This mapping should be updated whenever a user successfully calls `BriVault:joinEvent` and reset if the user withdraws or cancels their participation.

Additionally, a custom error (e.g., `AlreadyJoined`) should be defined and triggered in case the same address attempts to join again.

This solution prevents multiple registrations from the same address while keeping gas consumption minimal.

The `BriVault:cancelParticipation` function also requires a modification, since once an attacker cancels their participation, they receive the refund amount (minus the participation fee), but their address is **not removed** from the `usersAddress` array.

NOTE: It is still theoretically possible to attempt a Denial of Service (DoS) attack by repeatedly joining the event with multiple addresses. However, since each new `BriVault:joinEvent` call now requires a unique account and a separate deposit, the cost of performing such an attack becomes prohibitively high, making it economically infeasible in practice.

```

+ mapping (address => bool) hasJoined;

+ error alreadyJoined();
+ error notJoined();

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

+    if(hasJoined[msg.sender]){

```

```
+           revert alreadyJoined();
+       }

     if (countryId >= teams.length) {
         revert invalidCountry();
     }

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

+     hasJoined[msg.sender] = true;
     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);
 }

function cancelParticipation() public {
    if (block.timestamp >= eventStartDate) revert eventStarted();
+    if(!hasJoined[msg.sender]){
+        revert notJoined();
+    }

    uint256 refundAmount = stakedAsset[msg.sender];
    require(refundAmount > 0, "Nothing to refund");

    stakedAsset[msg.sender] = 0;

+    hasJoined[msg.sender] = false;
+    for (uint256 i = 0; i < usersAddress.length; i++) {
+        if (usersAddress[i] == msg.sender) {
+            usersAddress[i] = usersAddress[usersAddress.length - 1];
+            usersAddress.pop();
+            break;
+        }
    }
}
```

```
+         }
+     }
uint256 shares = balanceOf(msg.sender);
_burn(msg.sender, shares);
IERC20(asset()).safeTransfer(msg.sender, refundAmount);
}
```

[H-02] Inconsistent Share Ownership When Depositing to a Different Receiver**Description**

- The `BriVault::deposit` function accepts two parameters: `assets` (the deposit amount) and `receiver`. The `receiver` parameter allows a user to deposit tokens while assigning the resulting vault shares to a different address.
- The issue in the `BriVault::deposit` function occurs during the share minting process. Specifically, while the mapping `BriVault::stakedAsset` is correctly updated with the `BriVault::receiver` address, the minted shares are sent to `msg.sender` instead of the intended `BriVault::receiver`.

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

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

    uint256 fee = _getParticipationFee(assets);
    // charge on a percentage basis points
    if (minimumAmount + fee > assets) {
        revert lowFeeAndAmount();
    }

    uint256 stakeAsset = assets - fee;

    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 participantShares;
    }

```

Likelihood:

- This issue occurs every time a user calls `BriVault::deposit` with a `BriVault::receiver` address different from their own.

When the `BriVault::receiver` is the same as `msg.sender`, the behavior is correct; however, if they differ, the minted shares are incorrectly assigned to `msg.sender` instead of the specified `BriVault::receiver`.

Impact:

- This bug allows a receiver who does not actually hold any `BriVault:shares` to call the `BriVault:joinEvent` function, while preventing the user who originally called the `BriVault:deposit` function from participating — even though they do possess the `BriVault:shares`.
- When `BriVault:withdraw` is executed, part of the funds become irretrievable, since these `BriVault:shares` are excluded from the reward distribution calculation, leading to a permanent loss of assets.

Proof of Concept

Add this code snippet to the provided test file.

This test verifies the mismatch between the shares actually held and the mapping that tracks staked assets. It also demonstrates how a user who possesses shares cannot participate due to this mismatch, while a user who does not actually hold any shares can participate even though they should not be able to.

```

function testDepositSendSharesToWrongAddress() public {
    uint256 depositAmount = 1 ether;
    uint256 expectedShares = briVault.convertToShares(depositAmount) *
    ↵ (10000 - participationFeeBsp) / 10000;
    vm.startPrank(user1);
    mockToken.approve(address(briVault), depositAmount);
    briVault.deposit(depositAmount, user2);
    vm.stopPrank();
    // stakedAssetMapping should reflect user's shares amount
    assertEq(briVault.balanceOf(user1), expectedShares); // user1
    ↵ shares
}

```

```

        assertEquals(briVault.stakedAsset(user1), 0); // user1 mapping

        assertEquals(briVault.balanceOf(user2), 0); // user2 shares
        assertEquals(briVault.stakedAsset(user2), expectedShares ); // user2
→   mapping

        //User2 with no shares can join event
        vm.startPrank(user2);
        briVault.joinEvent(0); // joining with countryId 0
        vm.stopPrank();

        //User1 with shares can't join event
        vm.startPrank(user1);

→   vm.expectRevert(abi.encodeWithSelector(BriVault.noDeposit.selector));
        briVault.joinEvent(0); // joining with countryId 0
        vm.stopPrank();

    }
}

```

Recommended Mitigation

To prevent this issue, the `_to` parameter in the `BriVault::_mint` call should be replaced with `receiver` instead of `msg.sender`.

```

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

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

    uint256 fee = _getParticipationFee(assets);
    // charge on a percentage basis points
    if (minimumAmount + fee > assets) {
        revert lowFeeAndAmount();
    }

    uint256 stakeAsset = assets - fee;

    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);
+     _mint(receiver, participantShares);

    emit deposited (receiver, stakeAsset);

    return participantShares;
}
```

Medium**Low****Informational****Gas**