Protocol Audit Report

Prepared by: Bizarro

Table of Contents

Protocol Summary

AaveDIVAWrapper is a smart contract that acts as a **connector between DIVA Protocol and Aave V3**, allowing assets deposited into DIVA Protocol pools to **generate yield by supplying them on Aave V3**. The generated **yield is claimable by the owner** of the AaveDIVAWrapper contract.

The AaveDIVAWrapper contract was originally designed for DIVA Donate on Arbitrum, a parametric conditional donations platform, which aims to use the yield to purchase insurance policies to increase donation payouts beyond users' initial contributions. However, the contract can be utilized for any other use case enabled by DIVA Protocol (e.g., prediction markets, structured products, etc.).

Disclaimer

Bizarro found 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
	High	H	H/M	M
Likelihood	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.

Scope

The following contracts in `contracts/src/` are in scope:

```
src/
├── AaveDIVAWrapper.sol
├── AaveDIVAWrapperCore.sol
├── WToken.sol
├── interfaces
│   └── IAave.sol
│   └── IAaveDIVAWrapper.sol
│   └── IDIVA.sol
│   └── IWToken.sol
```

Roles

Executive Summary

Issues found

Severity	**Number of issues found**
High	1
Medium	2
Low	2
Info	0
Gas	0
Total	5

Findings

High

[H-1] `AaveDIVAWrapperCore::_redeemPositionToken` Violates Contract Invariants, Compromising Protocol Functionality

Description: The `AaveDIVAWrapperCore::_redeemPositionToken` calls the `DIVA` protocol's `redeemPositionToken` which burns the user's positionToken. https://github.com/Cyfrin/2025-01-diva/blob/23cdc88da7e2a9341f453854e876eee82a18e53e/contracts/src/AaveDIVAWrapperCore.sol#L294C9-L294C10

However, this operation burns only one token (`longToken` or `shortToken`) at a time, based on the `_positionToken` parameter.

Impact: This function will break the invariant `short token supply` = `long token supply` = `wToken supply`.

https://github.com/Cyfrin/2025-01-diva/blob/main/DOCUMENTATION.md#invariants

This disruption compromises the core functionality and balance of the protocol.

Proof of Code:

```
    it.only("Should Break The Invariant of the tokens", async() => {
      const beforeLongToken = await longTokenContract.totalSupply();
      const beforeShortToken = await shortTokenContract.totalSupply();
      const beforeWToken = await shortTokenContract.totalSupply();
      expect(beforeLongToken).to.be.eq(beforeShortToken);
      expect(beforeLongToken).to.be.eq(beforeWToken);
      expect(beforeShortToken).to.be.eq(beforeWToken);

      expect(longTokenBalance).to.be.gt(0);

      await s.aaveDIVAWrapper
        .connect(s.impersonatedSigner)
        .redeemPositionToken(
          poolParams.longToken,
          longTokenBalance,
          s.impersonatedSigner.address,
        );

      const afterLongToken = await longTokenContract.totalSupply();
      const afterShortToken = await shortTokenContract.totalSupply();
      const afterWToken = await shortTokenContract.totalSupply();
      expect(afterLongToken).not.to.be.eq(afterShortToken);
      expect(afterLongToken).not.to.be.eq(afterWToken);
      expect(afterShortToken).to.be.eq(afterWToken);
    })
```

Add This Test to the `redeemPositionToken` test group.

Recommended Mitigation: Refactor the functionality to ensure that the invariant (`short token supply` = `long token supply` = `wToken supply`) remains intact under all scenarios.

Medium

[M-1] Approving the maximum token amount `type(uint256).max` may not work for certain tokens that do not support this approval value.

Description: `AaveDIVAWrapperCore::_registerCollateralToken` approves the maximum amount of collateralToken to be spent by the `aaveV3Pool`. The tokens like `UNI` or `COMP` will revert when approving the max tokens.

https://github.com/Uniswap/governance/blob/eabd8c71ad01f61fb54ed6945162021ee419998e/contracts/Uni.sol#L149

```
 function approve(address spender, uint rawAmount) external returns (bool)
{
        uint96 amount;
        if (rawAmount == uint(-1)) {
            amount = uint96(-1);
        } else {
            amount = safe96(rawAmount, "Uni::approve: amount exceeds 96
bits");
        }

        allowances[msg.sender][spender] = amount;

        emit Approval(msg.sender, spender, amount);
        return true;
    }
```

Both the tokens contain this piece of code which reverts when the amount exceeds 96 bits.

Impact: This functionality can revert for tokens that doesn't suppport max `unit256` approval, If this occurs, the entire batch of tokens awaiting approval will also fail.

Proof of Concept: https://github.com/Cyfrin/2025-01-diva/blob/5b7473c13adf54a4cd1fd6b0f37ab6529c4487dc/contracts/src/AaveDIVAWrapperCore.sol#L116

Recommended Mitigation: I would suggest approve only the necessay amount of token to the `aaveV3Pool` instead of the `type(uint256).max` amount.

[M-2] Some ERC20 tokens revert when transfer amount is zero which may affect the functionality of the `batchAddLiquidity` function.

Description: In `AaveDIVAWrapperCore::_addLiquidity` function safeTransfers collateralToken from the msg.sender to the protocol address to provide liquidity to the aaveLiquidityPool, but the _addLiquidity function doesn't check the _collateralAmount if it's zero or not.

Impact: Some ERC20 tokens like such as `LEND` will revert if this is attempted, which may cause transactions that involve other tokens (such as batch operations) to fully revert.

Proof of Concept: Proof of concept

Recommended Mitigation: Add a zero Amount check in the beginning of the function.

```
function _addLiquidity(
        bytes32 _poolId,
        uint256 _collateralAmount,
        address _longRecipient,
        address _shortRecipient
    ) internal {
+       require(_collateralAmount !=0, "Invalid Collateral Amount");
        // Verify that the collateral token used in the DIVA Protocol pool
corresponds to a registered
        // collateral token in the AaveDIVAWrapper contract. Returns zero
address if the wToken is not registered.
        IDIVA.Pool memory _pool = IDIVA(_diva).getPoolParameters(_poolId);
        address _collateralToken =
_wTokenToCollateralToken[_pool.collateralToken];
        ......
        ...
    }
```

Low

[L-1] `AaveDIVAWrapperCore::_redeemToken` function is not necessary for this contract, because the wTokens are only minted to the wrapperContract and user does not own any wToken themselves.

Description: `AaveDIVAWrapperCore::_redeemToken` function Burns the wToken from the userBalance, but in this contract the user does not own any wTokens as all the tokens are minted to the wrapper contract.

Impact: This function will never execute because the owner does not own any wTokens. And if the owner decides to mint more wTokens without supplying the collateral tokens to Aave, the invariant will break and owner won't be able to claim the yield.

Proof of Concept: wTokens are minted only to the wrapper contract.

Here is the Invariant Break.

Recommended Mitigation: Remove this function from the contract.

[L-2] User's shortToken and longToken balance should be checked before transferring the tokens to the wrapper contract in `AaveDIVAWrapperCore::_removeLiquidity`.

Description: The `AaveDIVAWrapperCore::_removeLiquidity` transfers the short and long token to the wrapper contract without checking the user's token balance.

Impact: unwanted revert with no proper message. https://github.com/Cyfrin/2025-01-diva/blob/main/DOCUMENTATION.md#reverts-2

Proof of Concept: POC

Recommended Mitigation:

```
  function _removeLiquidity(
         bytes32 _poolId,
         uint256 _positionTokenAmount,
         address _recipient
     ) internal returns (uint256) {
         ...

         uint256 _userBalanceShort =
_shortTokenContract.balanceOf(msg.sender);
         uint256 _userBalanceLong =
_longTokenContract.balanceOf(msg.sender);
         uint256 _positionTokenAmountToRemove = _positionTokenAmount;
+         if(_positionTokenAmount > _userBalanceLong ||
_positionTokenAmount > _userBalanceShort){
+             revert NOT_ENOUGH_TOKEN_BALANCE();
+         }
         if (_positionTokenAmount == type(uint256).max) {
             _positionTokenAmountToRemove = _userBalanceShort >
_userBalanceLong ? _userBalanceLong : _userBalanceShort;
         }

         ...
     }
```