Protocol Audit Report

Version 1.0

Moon.io

January 2, 2026

Vault Guardian Audit Report

Moon.io

Jan 2, 2026

Prepared by: Moon.io

Lead Auditors: Divyansh (ELO_Anxiety)

Table of Contents

Protocol Summary

This protocol allows users to deposit certain ERC20s into an ERC4626 vault managed by a human being, or a `vaultGuardian`. The goal of a `vaultGuardian` is to manage the vault in a way that maximizes the value of the vault for the users who have despoited money into the vault.

You can think of a `vaultGuardian` as a fund manager.

To prevent a vault guardian from running off with the funds, the vault guardians are only allowed to deposit and withdraw the ERC20s into specific protocols.

- Aave v3
- Uniswap v2
- None (just hold)

These 2 protocols (plus "none" makes 3) are known as the "investable universe".

The guardian can move funds in and out of these protocols as much as they like, but they cannot move funds to any other address.

The goal of a vault guardian, is to move funds in and out of these protocols to gain the most yield. Vault guardians charge a performance fee, the better the guardians do, the larger fee they will earn.

Anyone can become a Vault Guardian by depositing a certain amount of the ERC20 into the vault. This is called the `guardian stake`. If a guardian wishes to stop being a guardian, they give out all user deposits and withdraw their guardian stake.

Users can then move their funds between vault managers as they see fit.

The protocol is upgradeable so that if any of the platforms in the investable universe change, or we want to add more, we can do so.

User flow

1. User deposits an ERC20 into a guardian's vault
2. The guardian automatically move the funds based on their strategy
3. The guardian can update the settings of their strategy at any time and move the funds
4. To leave the pool, a user just calls `redeem` or `withdraw`

The DAO

Guardians can earn DAO tokens by becoming guardians. The DAO is responsible for: - Updating pricing parameters - Getting a cut of all performance of all guardians

Summary

Users can stake some ERC20s to become a vault guardian. Other users can allocate them funds in order to maximize yield. The guardians can move the funds between Uniswap, Aave, or just hold the funds. The guardians are incentivized to maximize yield, as they earn a performance fee.

Disclaimer

The Moon 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
	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.

Audit Details

Quickstart

```
git clone https://github.com/Cyfrin/8-vault-guardians-audit
cd 8-vault-guardians-audit
make
```

Optional Gitpod

If you can't or don't want to run and install locally, you can work with this repo in Gitpod. If you do this, you can skip the `clone this repo` part.

Usage

Testing

Set the `RPC_URL_MAINNET` environment variable with the URL of a mainnet RPC node. It's used for tests that fork Ethereum mainnet state.

Then run:

```
1 forge test
```

Test Coverage

```
1 forge coverage
```

and for coverage based testing:

```
1 forge coverage --report debug
```

Misc

- Art made with asciiart.eu
- headers from t11/headers

Scope

- Commit Hash: xx
- In Scope:

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- Tokens:
  - weth: https://etherscan.io/token/0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2
  - link: https://etherscan.io/token/0x514910771af9ca656af840dff83e8264ecf986ca
  - usdc: https://etherscan.io/token/0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48

Roles

- Liquidity Providers: Users who have liquidity deposited into the pools. Their shares are represented by the LP ERC20 tokens. They gain a 0.3% fee every time a swap is made.
- Users: Users who want to swap tokens.

Executive Summary

Issues found

Severity	Number of issues found
High	8
Medium	0
Low	2
Info/Gas	1
Total	11

Findings

HIGH

[H-1] Incorrect Initialization of Uniswap Pair token inside the contructor of `VaultShares` contract can lead to zero address initialization for `i_uniswapLiquidityToken`.

Description: The constructor of the Contract `VaultShares` is having a crucial error which is as follows :

```
    .
    .
    constructor(ConstructorData memory constructorData)
        ERC4626(constructorData.asset)
        ERC20(constructorData.vaultName, constructorData.vaultSymbol)
        AaveAdapter(constructorData.aavePool)
        UniswapAdapter(constructorData.uniswapRouter, constructorData.
            weth, constructorData.usdc)
    {
        i_guardian = constructorData.guardian;
        i_guardianAndDaoCut = constructorData.guardianAndDaoCut;
        i_vaultGuardians = constructorData.vaultGuardians;
        s_isActive = true;
        updateHoldingAllocation(constructorData.allocationData);//msg.
            sender must be vaultGuardian

        // External calls
        i_aaveAToken =
            IERC20(IPool(constructorData.aavePool).getReserveData(
                address(constructorData.asset)).aTokenAddress);
@>      i_uniswapLiquidityToken = IERC20(i_uniswapFactory.getPair(
    address(constructorData.asset), address(i_weth)));// @audit-high
    what if the token itself is weth ???
    }
    .
    .
```

The function `getPair` described in the `UniswapV2Factory`(Link to Original contract) is as follows :

```
    mapping(address => mapping(address => address)) public getPair;
```

This shows that the `getPair` function will return `address(0)` if both the addresses passed in the function are same as no address exists for (weth,weth) pool.

Impact: This is breaking the uniswap investment logic and is dangerous.

Proof of Concept: 1. A user calls the `becomeGuardian(AllocationData)` function (for WETH vault). 2. `i_uniswapLiquidityToken` is set to `address(0)`.

PoC

1. Insert this interface in `test/fork/WethFork.t.sol` after the imports

```
    .
    .
    import {Fork_Test} from "./Fork.t.sol";

    interface IWETH is IERC20 {
```

```
        function deposit() external payable;
        function withdraw(uint256 wad) external;
    }
    .
    .
```

2. Insert this test in the same file

```
    function testbecomeGuardian() public{
        vm.deal(guardian,mintAmount);

        vm.startPrank(guardian);
        IWETH(address(weth)).deposit{value:mintAmount}();//mintAmount
            =100 ether
        weth.approve(address(vaultGuardians), mintAmount);
        address wethVault = vaultGuardians.becomeGuardian(
            allocationData);
        wethVaultShares = VaultShares(wethVault);
        vm.stopPrank();

        console.log("uniswap liquidity token address",wethVaultShares.
            getUniswapLiquidtyToken());
        vm.assertEq(wethVaultShares.getUniswapLiquidtyToken(),address
            (0));
    }
```

3. Paste this command in the terminal

```
 forge test --mt testbecomeGuardian --rpc-url $RPC_URL_MAINNET -vvv
```

RESULT OF THE TEST -

```

Ran 1 test for test/fork/WethFork.t.sol:WethForkTest
[PASS] testbecomeGuardian() (gas: 4583708)
Logs:
  uniswap liquidity token address 0
      x0000000000000000000000000000000000000000

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 5.05s
    (18.05ms CPU time)

Ran 1 test suite in 7.20s (5.05s CPU time): 1 tests passed, 0 failed, 0
    skipped (1 total tests)
```

Recommended Mitigation: Add this check in the constructor and change the visibility of the variable:

```
      .
      .
-     IERC20 internal immutable i_uniswapLiquidityToken;
+     IERC20 internal s_uniswapLiquidityToken;
      .
      .
      constructor(ConstructorData memory constructorData)
          ERC4626(constructorData.asset)
          ERC20(constructorData.vaultName, constructorData.vaultSymbol)
          AaveAdapter(constructorData.aavePool)
          UniswapAdapter(constructorData.uniswapRouter, constructorData.
              weth, constructorData.usdc)
      {
          i_guardian = constructorData.guardian;
          i_guardianAndDaoCut = constructorData.guardianAndDaoCut;
          i_vaultGuardians = constructorData.vaultGuardians;
          s_isActive = true;
          updateHoldingAllocation(constructorData.allocationData);//msg.
              sender must be vaultGuardian

          // External calls
          i_aaveAToken =
              IERC20(IPool(constructorData.aavePool).getReserveData(
                  address(constructorData.asset)).aTokenAddress);
+         if(address(constructorData.asset)==address(i_weth)){
+             s_uniswapLiquidityToken = IERC20(i_uniswapFactory.getPair(
      address(constructorData.asset), address(i_tokenOne)));
+         }else{
+             s_uniswapLiquidityToken = IERC20(i_uniswapFactory.getPair(
      address(constructorData.asset), address(i_weth)));
+         }
-         i_uniswapLiquidityToken = IERC20(i_uniswapFactory.getPair(
      address(constructorData.asset), address(i_weth)));// @audit-high
      what if the token itself is weth ???
      }
```

[H-2] Slippage is not specified in the `UniswapAdapter::_uniswapInvest` function, hence will lead to MEV attack .

Description: In the `_uniswapInvest` function, at two places minimum amount is not specified which can lead to MEV attack . Also the deadline is block.timestamp which means it can be any block, hence can lead to frontrunning issues .

Found instances -

```
          //@audit mev
          uint256[] memory amounts = i_uniswapRouter.
              swapExactTokensForTokens({
```

```
3                 amountIn: amountOfTokenToSwap,
4   @>            amountOutMin: 0,
5                 path: s_pathArray,
6                 to: address(this),
7   @>            deadline: block.timestamp
8           });
9
10          succ = counterPartyToken.approve(address(i_uniswapRouter),
                amounts[1]);
11          if (!succ) {
12              revert UniswapAdapter__TransferFailed();
13          }
14          //q we are give permission two 2x tokens of weth if token is
                weth
15          succ = token.approve(address(i_uniswapRouter),
                amountOfTokenToSwap + amounts[0]);
16          if (!succ) {
17              revert UniswapAdapter__TransferFailed();
18          }
19
20          // amounts[1] should be the WETH amount we got back
21          // @audit amountAMin and amountBmin should not be zero ITS MEV
                !!
22          (uint256 tokenAmount, uint256 counterPartyTokenAmount, uint256
                liquidity) = i_uniswapRouter.addLiquidity({
23              tokenA: address(token),
24              tokenB: address(counterPartyToken),
25              amountADesired: amountOfTokenToSwap + amounts[0],
26              amountBDesired: amounts[1],
27  @>          amountAMin: 0,
28  @>          amountBMin: 0,
29              to: address(this),
30  @>          deadline: block.timestamp
31          });
```

Impact: :HIGH

Likelihood :HIGH

Proof of Concept: - Suppose the weth/usdc pool has 100/200000 tokens respectively - Some MEV bot will see this and take a flash loan from aave of 100,000 USDC and swapped with WETH - 1 WETH»2000USDC now - The user wants to get weth in exchange of USDC - The user hence gets less weth for the usdc - Now the user deposits the usdc and the equivalent amount of weth from the vault(weth will be remaining there in the pool ,but all the usdc will be invested) - Bot now dumps all the WETH and gets USDC back with some profit . - Loan is repaid back and user suffered some loss.

Recommended Mitigation: It is recommenened to specify the minimum amount based on the amount invested. Also it is suggested to change the deadline to some meaningful deadline.

[H-3] Slippage is not specified in the `UniswapAdapter::_uniswapDivest` function, hence will lead to MEV attack .

Description: In the `_uniswapInvest` function, at two places minimum amount is not specified which can lead to MEV attack . Also the deadline is block.timestamp which means it can be any block, hence can lead to frontrunning issues .

Found instances -

```
1           //@audit MEV!!
2           (uint256 tokenAmount, uint256 counterPartyTokenAmount) =
               i_uniswapRouter.removeLiquidity({
3               tokenA: address(token),
4               tokenB: address(counterPartyToken),
5               liquidity: liquidityAmount,
6   @>          amountAMin: 0,
7   @>          amountBMin: 0,
8               to: address(this),
9   @>          deadline: block.timestamp
10          });
11          s_pathArray = [address(counterPartyToken), address(token)];
12          //@audit MEV
13          uint256[] memory amounts = i_uniswapRouter.
               swapExactTokensForTokens({
14              amountIn: counterPartyTokenAmount,
15  @>          amountOutMin: 0,
16              path: s_pathArray,
17              to: address(this),
18  @>          deadline: block.timestamp
19          });
```

Impact: :HIGH

Likelihood :HIGH

Recommended Mitigation: It is recommenened to specify the minimum amount based on the `liquidityAmount`. Also it is suggested to change the deadline to some meaningful deadline.

[H-4] No token decimal check inside `VaultGuardianBase::_becomeTokenGuardian` makes it almost impossible for anyone to become guardian of any non-weth token due to extremly high staking amount.

Description: In the `_becomeTokenGuardian` function, there is no check regarding token decimal, which makes it impossible for anyone to become a `guardian` for a non-weth token due to extremly high staking amount. Hence, breaking the whole protocol.

Vulnerable Code:

```
1      function _becomeTokenGuardian(IERC20 token, VaultShares tokenVault)
           private returns (address) {
2          s_guardians[msg.sender][token] = IVaultShares(address(
               tokenVault));
3          emit GuardianAdded(msg.sender, token);
4          i_vgToken.mint(msg.sender, s_guardianStakePrice);
5          //@audit high this is no good actually
6  @>      token.safeTransferFrom(msg.sender, address(this),
       s_guardianStakePrice);// stake guardian is putting in this contract
       10 weth , so if the token is usdc here , where the decimals is 6, 10
        ether means 10^19/10^6 =10^13 usdc
7          bool succ = token.approve(address(tokenVault),
               s_guardianStakePrice);
8          if (!succ) {
9              revert VaultGuardiansBase__TransferFailed();
10         }
11         uint256 shares = tokenVault.deposit(s_guardianStakePrice, msg.
               sender);// putting the 10 weth in the vault
12         if (shares == 0) {
13             revert VaultGuardiansBase__TransferFailed();
14         }
15         return address(tokenVault);
16     }
```

Impact: The protocol can't have any vault other than `WETH`, hence it's a HIGH impact **Likelihood** HIGH likelihood as any weht guardian will try to become guardian of another token other than WETH.

Proof of Concept: Following is the proof of concept: 1. A user become a guardian of `WETH` vault by staking `10 ether` WETH. 2. The user than tries to become guardian of `USDC` vault and calls the `becomeTokenGuardian` function which internally calls `_becomeTokenGuardian`. 3. This function again wants the user to stake `10 ether` worth `usdc` which is actually 10*(10* ***18)/(10*** *6) => 10* (10 ** 8) => `10^9 usdc`, which is a way too big amount and the user will probably not have that much. 4. The function will revert as the user can't transfer that.

PoC

1. Paste these lines in the `WethFork.t.sol`

```
1      .
2      .
3      address public WHALE=0x47ac0Fb4F2D84898e4D9E7b4DaB3C24507a6D503;
4      .
5      .
6      modifier hasGuardian() {
7          vm.deal(guardian,mintAmount);
8
9          vm.startPrank(guardian);
10         IWETH(address(weth)).deposit{value:mintAmount}();//mintAmount
```

```
            =100 ether
11          weth.approve(address(vaultGuardians), type(uint256).max);
12          address wethVault = vaultGuardians.becomeGuardian(
               allocationData);
13          wethVaultShares = VaultShares(wethVault);
14          vm.stopPrank();
15          _;
16      }
```

2. Paste this test

```
1       function test_becomeTokenGuardianIsHavingDecimalError() public
           hasGuardian{
2           vm.startPrank(WHALE);
3           usdc.transfer(guardian,usdcMint);
4           console.log("balance of user usdc:",usdc.balanceOf(guardian));
5           vm.stopPrank();
6
7           vm.startPrank(guardian);
8           usdc.approve(address(vaultGuardians), type(uint256).max);
9           vm.expectRevert();// due to 1 e14 usdc has to be send which is
               way too much
10          vaultGuardians.becomeTokenGuardian(allocationData,usdc);
11
12          vm.assertEq(usdcMint,usdc.balanceOf(guardian));
13          vm.stopPrank();
14      }
```

Recommended Mitigation:

```
1       function _becomeTokenGuardian(IERC20 token, VaultShares tokenVault)
            private returns (address) {
2           s_guardians[msg.sender][token] = IVaultShares(address(
               tokenVault));
3           emit GuardianAdded(msg.sender, token);
4           i_vgToken.mint(msg.sender, s_guardianStakePrice);
5           //@audit high this is no good actually
6   -       token.safeTransferFrom(msg.sender, address(this),
        s_guardianStakePrice);
7   +       if(token==weth){
8   +           token.safeTransferFrom(msg.sender, address(this),
        s_guardianStakePrice);
9   +       }
10  +       token.safeTransferFrom(msg.sender,address(this),10_000*(token.
        decimals())) //10_000 for usdc and 12*10_000 for link
11          bool succ = token.approve(address(tokenVault),
               s_guardianStakePrice);
12          if (!succ) {
13              revert VaultGuardiansBase__TransferFailed();
14          }
```

```
15          uint256 shares = tokenVault.deposit(s_guardianStakePrice, msg.
                sender);// putting the 10 weth in the vault
16          if (shares == 0) {
17              revert VaultGuardiansBase__TransferFailed();
18          }
19          return address(tokenVault);
20      }
```

[H-5] Incorrect share calculation in `VaultShares::deposit` making the calculations inccorect and leading to loss of funds.

Description: The shares in the `deposit` function is not getting distributed correctly, as the funds which are being invested are not considered while calculating the shares, hence leads to more shares getting minted than expected .

Vulnerable Code:

```
1       function deposit(uint256 assets, address receiver)
2           public
3           override(ERC4626, IERC4626)
4           isActive
5           nonReentrant
6           returns (uint256)
7       {
8           if (assets > maxDeposit(receiver)) {
9               revert VaultShares__DepositMoreThanMax(assets, maxDeposit(
                    receiver));
10          }
11
12          uint256 shares = previewDeposit(assets);
13          _deposit(_msgSender(), receiver, assets, shares);
14
15          //q maybe this can cause some problem
16          _mint(i_guardian, shares / i_guardianAndDaoCut);
17          _mint(i_vaultGuardians, shares / i_guardianAndDaoCut);
18
19  @>      _investFunds(assets);
20          return shares;
21      }
```

Impact: User gets higher share for the same amount and hence can claim more tokens ,leading to Mismanagement of ETH.

Proof of Concept: 1. The vault guardian creates a new Vault and set the allocation data as (500,250,250) and deposits 10e18 and gets back 10e18 + 1e16 shares and the maoin contract gets 1e16 shares. 2. User comes and deposits 10e18 3. Shares which the user will receieve = `10e18*(10.02e18+1)/(5e18+x)` //5e18 because 500 is holdAllocation 3. Here x is the small amount

which is left when investing in uniswap pairs. 4. suppose x is 1e16 5. user gets almost 20.4e18 shares 6. Both user and guardian have invested deposited same amount but the user has got more shares than the user.

PoC

```
    function testDepositFunctionIsVulnerable() public {
        vm.deal(guardian,mintAmount);
        vm.deal(user,mintAmount);

        vm.startPrank(guardian);
        IWETH(address(weth)).deposit{value:mintAmount}();//mintAmount
            =100 ether
        weth.approve(address(vaultGuardians), type(uint256).max);
        console.log("balance of guardian:",weth.balanceOf(guardian));

        address wethVault = vaultGuardians.becomeGuardian(
            allocationData);//(500,250,250)
        wethVaultShares = VaultShares(wethVault);
        vm.stopPrank();

        // vm.warp(block.timestamp+2 days);

        console.log("totalSupplyYet:",wethVaultShares.totalSupply());//
            5e18
        console.log("total Assets:",wethVaultShares.totalAssets());
        console.log("weth balance in the vault:",weth.balanceOf(address
            (wethVaultShares)));//5e18
        assertEq(wethVaultShares.totalSupply(),10e18+2e16);

        vm.startPrank(user);
        IWETH(address(weth)).deposit{value:mintAmount}();//mintAmount
            =100 ether
        weth.approve(address(wethVaultShares), type(uint256).max);
        console.log("balance of user:",weth.balanceOf(user));

        uint256 sharesRecievedByUser=wethVaultShares.deposit(10e18,user
            );
        console.log("shares which the users got received: ",
            sharesRecievedByUser);

        assert(sharesRecievedByUser<=20e18+4e16+2);// max shares user
            can get when x is zero
        assert(wethVaultShares.balanceOf(user)>wethVaultShares.
            balanceOf(guardian));

        vm.stopPrank();
    }
```

Recommended Mitigation: Consider redesigning the process or maybe add `divestThanInvest`

modifier in this .

[H-6] Uniswap Liquidity Token is not approved, leading to the revert of `_uniswapDivest` due to the revert of the function `UniswapRouter::removeLiquidity`.

Description: The `uniswapLiquidityToken` which is being minted to the vault after invest in uniswap any two token pairs are not approved by the vault, hence leading to revert everytime the modifier `divestThanInvest` is called due to the revert of `UniswapRouter::removeLiquidity` inside the `_uniswapDivest`.

Code:

- `divestThanInvest` modifier

```
1      modifier divestThenInvest() {
2          uint256 uniswapLiquidityTokensBalance = s_uniswapLiquidityToken
               .balanceOf(address(this));
3          uint256 aaveAtokensBalance = i_aaveAToken.balanceOf(address(
               this));
4
5          // Divest
6          if (uniswapLiquidityTokensBalance > 0) {
7              // @audit approval needed
8  @>          // s_uniswapLiquidityToken.approve(address(i_uniswapRouter)
       ,uniswapLiquidityTokensBalance);
9              _uniswapDivest(IERC20(asset()),
                   uniswapLiquidityTokensBalance);
10         }
11         if (aaveAtokensBalance > 0) {
12             _aaveDivest(IERC20(asset()), aaveAtokensBalance);
13         }
14
15         _;
16
17         // Reinvest
18         if (s_isActive) {
19             _investFunds(IERC20(asset()).balanceOf(address(this)));
20         }
21     }
```

Impact: The function will always revert ,hence there is no way for a user or anyone to redeem the assets which are invested.

Recommended Mitigation: Make this approval here :

```
1      modifier divestThenInvest() {
2          uint256 uniswapLiquidityTokensBalance = s_uniswapLiquidityToken
               .balanceOf(address(this));
```

```
3           uint256 aaveAtokensBalance = i_aaveAToken.balanceOf(address(
                this));
4
5           // Divest
6           if (uniswapLiquidityTokensBalance > 0) {
7               // @audit approval needed
8   +           s_uniswapLiquidityToken.approve(address(i_uniswapRouter),
        uniswapLiquidityTokensBalance);
9               _uniswapDivest(IERC20(asset()),
                    uniswapLiquidityTokensBalance);
10          }
11          if (aaveAtokensBalance > 0) {
12              _aaveDivest(IERC20(asset()), aaveAtokensBalance);
13          }
14
15          _;
16
17          // Reinvest
18          if (s_isActive) {
19              _investFunds(IERC20(asset()).balanceOf(address(this)));
20          }
21      }
```

[H-7] Counterparty Token is not approved, leading to the revert of `_uniswapDivest` due to the revert of the function `UniswapRouter::swapExactTokensForTokens`.

Description: The `counterpartyToken` is not approved by the vault, hence leading to revert everytime the modifier `divestThanInvest` is called due to the revert of `UniswapRouter::swapExactTokensForTokens` inside the `_uniswapDivest`.

Code :

```
1       function _uniswapDivest(IERC20 token, uint256 liquidityAmount)
            internal returns (uint256 amountOfAssetReturned) {
2           IERC20 counterPartyToken = token == i_weth ? i_tokenOne :
                i_weth;
3           //@written MEV!!
4           (uint256 tokenAmount, uint256 counterPartyTokenAmount) =
                i_uniswapRouter.removeLiquidity({
5               tokenA: address(token),
6               tokenB: address(counterPartyToken),
7               liquidity: liquidityAmount,
8               amountAMin: 0,
9               amountBMin: 0,
10              to: address(this),
11              deadline: block.timestamp
12          });
13          s_pathArray = [address(counterPartyToken), address(token)];
```

```
@>      //@audit approval needed
@>      // counterPartyToken.approve(address(i_uniswapRouter),
    counterPartyTokenAmount);
        //@written MEV
        uint256[] memory amounts = i_uniswapRouter.
            swapExactTokensForTokens({
            amountIn: counterPartyTokenAmount,
            amountOutMin: 0,
            path: s_pathArray,
            to: address(this),
            deadline: block.timestamp
        });
        emit UniswapDivested(tokenAmount, amounts[1]);
        amountOfAssetReturned = amounts[1];
    }
```

Impact: The function will always revert ,hence there is no way for a user or anyone to redeem the assets which are invested.

Recommended Mitigation: Make this approval here :

```
    function _uniswapDivest(IERC20 token, uint256 liquidityAmount)
        internal returns (uint256 amountOfAssetReturned) {
        IERC20 counterPartyToken = token == i_weth ? i_tokenOne :
            i_weth;
        //@written MEV!!
        (uint256 tokenAmount, uint256 counterPartyTokenAmount) =
            i_uniswapRouter.removeLiquidity({
            tokenA: address(token),
            tokenB: address(counterPartyToken),
            liquidity: liquidityAmount,
            amountAMin: 0,
            amountBMin: 0,
            to: address(this),
            deadline: block.timestamp
        });
        s_pathArray = [address(counterPartyToken), address(token)];

+       counterPartyToken.approve(address(i_uniswapRouter),
    counterPartyTokenAmount);

        uint256[] memory amounts = i_uniswapRouter.
            swapExactTokensForTokens({
            amountIn: counterPartyTokenAmount,
            amountOutMin: 0,
            path: s_pathArray,
            to: address(this),
            deadline: block.timestamp
        });
        emit UniswapDivested(tokenAmount, amounts[1]);
        amountOfAssetReturned = amounts[1];
```

```
26      }
```

[H-8] A malicious guardian can mint infinite Governance Tokens, leading the guardian to take over the DAO and hence destroying the protocol.

Description: In the `VaultGuardianBase::_quitGuardian` function, the governance tokens are not burned, hence any guardian can mint infinite number of `governanceTokens` by repeatedly calling `becomeGuardian` and `quitGuardian`

Code:

```
1     //q governance tokens are not burnt ??isn't that dangerous to the
          dao ??
2     function _quitGuardian(IERC20 token) private returns (uint256) {
3         IVaultShares tokenVault = IVaultShares(s_guardians[msg.sender][
              token]);
4         s_guardians[msg.sender][token] = IVaultShares(address(0));
5         emit GaurdianRemoved(msg.sender, token);
6         tokenVault.setNotActive();
7         uint256 maxRedeemable = tokenVault.maxRedeem(msg.sender);
8         uint256 numberOfAssetsReturned = tokenVault.redeem(
              maxRedeemable, msg.sender, msg.sender);
9         return numberOfAssetsReturned;
10    }
```

Impact: HIGH

Proof of Concept: 1. User calls `becomeGuardian` and in the same transaction calls `quitGuardian` . 2. This can be done multiple time and then a proposal can be called which will give the guardian power over the DAO and hence can change the important state variables.

Recommended Mitigation: Consider burning the governance token in the `_quitGuardian` function to avoid this .

LOW

[L-1] Return value `amountOfAssetReturned` is not used in the `AaveAdapter::_aaveDivest` function, leading to compiler warning and lack to information due to unused return value

Description: In the `_aaveDivest` function , `uint256 amountOfAssetReturned` is unused ,therefore leading to compiler warnings .

```
@>  function _aaveDivest(IERC20 token, uint256 amount) internal returns
    (uint256 amountOfAssetReturned) {
        i_aavePool.withdraw({
            asset: address(token),
            amount: amount,
            to: address(this)
        });
    }
```

Recommended Mitigation:

```
-   function _aaveDivest(IERC20 token, uint256 amount) internal returns
    (uint256 amountOfAssetReturned) {
+   function _aaveDivest(IERC20 token, uint256 amount) internal returns
    (uint256 /*amountOfAssetReturned*/)
-       i_aavePool.withdraw({
+       return i_aavePool.withdraw({
            asset: address(token),
            amount: amount,
            to: address(this)
        });
    }
```

[L-2] Incorrect Token name and Token symbol in `VaultGuardianBase::becomeTokenGuardian` when token provided is `LINK`.

Description: In the `becomeTokenGuardian` function, when creating a new `VaultShares` contract, vault name and vault symbol is incorrect when someone tries to make a new Vault for `LINK` tokens.

```
function becomeTokenGuardian(AllocationData memory allocationData,
    IERC20 token)
        external
        onlyGuardian(i_weth)
        returns (address)
    {
        //slither-disable-next-line uninitialized-local
        VaultShares tokenVault;
        if (address(token) == address(i_tokenOne)) {
            tokenVault =
            new VaultShares(IVaultShares.ConstructorData({
                asset: token,
                vaultName: TOKEN_ONE_VAULT_NAME,
                vaultSymbol: TOKEN_ONE_VAULT_SYMBOL,
                guardian: msg.sender,
                allocationData: allocationData,
                aavePool: i_aavePool,
```

```
                uniswapRouter: i_uniswapV2Router,
                guardianAndDaoCut: s_guardianAndDaoCut,
                vaultGuardians: address(this),
                weth: address(i_weth),
                usdc: address(i_tokenOne)
            }));
        } else if (address(token) == address(i_tokenTwo)) {
            tokenVault =
            new VaultShares(IVaultShares.ConstructorData({
                asset: token,
@>              vaultName: TOKEN_ONE_VAULT_NAME,//@audit isn't this be TOKEN_TWO_VAULT_NAME
@>              vaultSymbol: TOKEN_ONE_VAULT_SYMBOL,//@audit isn't this be TOKEN_TWO_VAULT_SYMBOL
                guardian: msg.sender,
                allocationData: allocationData,
                aavePool: i_aavePool,
                uniswapRouter: i_uniswapV2Router,
                guardianAndDaoCut: s_guardianAndDaoCut,
                vaultGuardians: address(this),
                weth: address(i_weth),
                usdc: address(i_tokenOne)
            }));
        } else {
            revert VaultGuardiansBase__NotApprovedToken(address(token));
        }
        return _becomeTokenGuardian(token, tokenVault);
    }
```

Impact: LOW

Recommended Mitigation:

```
    function becomeTokenGuardian(AllocationData memory allocationData, IERC20 token)
        external
        onlyGuardian(i_weth)
        returns (address)
    {
        //slither-disable-next-line uninitialized-local
        VaultShares tokenVault;
        if (address(token) == address(i_tokenOne)) {
            tokenVault =
            new VaultShares(IVaultShares.ConstructorData({
                asset: token,
                vaultName: TOKEN_ONE_VAULT_NAME,
                vaultSymbol: TOKEN_ONE_VAULT_SYMBOL,
                guardian: msg.sender,
                allocationData: allocationData,
                aavePool: i_aavePool,
```

```
                uniswapRouter: i_uniswapV2Router,
                guardianAndDaoCut: s_guardianAndDaoCut,
                vaultGuardians: address(this),
                weth: address(i_weth),
                usdc: address(i_tokenOne)
            }));
        } else if (address(token) == address(i_tokenTwo)) {
            tokenVault =
            new VaultShares(IVaultShares.ConstructorData({
                asset: token,
-               vaultName: TOKEN_ONE_VAULT_NAME,//@audit isn't this be TOKEN_TWO_VAULT_NAME
-               vaultSymbol: TOKEN_ONE_VAULT_SYMBOL,//@audit isn't this be TOKEN_TWO_VAULT_SYMBOL
+               vaultName: TOKEN_TWO_VAULT_NAME,
+               vaultSymbol: TOKEN_TWO_VAULT_SYMBOL,
                guardian: msg.sender,
                allocationData: allocationData,
                aavePool: i_aavePool,
                uniswapRouter: i_uniswapV2Router,
                guardianAndDaoCut: s_guardianAndDaoCut,
                vaultGuardians: address(this),
                weth: address(i_weth),
                usdc: address(i_tokenOne)
            }));
        } else {
            revert VaultGuardiansBase__NotApprovedToken(address(token));
        }
        return _becomeTokenGuardian(token, tokenVault);
    }
```

Gas/Informational

[G-1] Unused variables, events and errors in `VaultGuardianBase.sol` file, make the deployment more costly

Description

```
@>  error VaultGuardiansBase__NotEnoughWeth(uint256 amount, uint256 amountNeeded);//@audit low not used error
    error VaultGuardiansBase__NotAGuardian(address guardianAddress, IERC20 token);
@>  error VaultGuardiansBase__CantQuitGuardianWithNonWethVaults(address guardianAddress);// @audit low unused error
    error VaultGuardiansBase__CantQuitWethWithThisFunction();
    error VaultGuardiansBase__TransferFailed();
```

```
6  @>  error VaultGuardiansBase__FeeTooSmall(uint256 fee, uint256
       requiredFee);//@audit low unused error
7      error VaultGuardiansBase__NotApprovedToken(address token);
8      .
9      .
10 @>  uint256 private constant GUARDIAN_FEE = 0.1 ether;// @audit low not
       used anywhere
11     .
12     .
13     event GuardianAdded(address guardianAddress, IERC20 token);
14     event GaurdianRemoved(address guardianAddress, IERC20 token);
15 @>  event InvestedInGuardian(address guardianAddress, IERC20 token,
       uint256 amount);//@audit low not used
16 @>  event DinvestedFromGuardian(address guardianAddress, IERC20 token,
       uint256 amount);//@audit low not used function
17     event GuardianUpdatedHoldingAllocation(address guardianAddress,
           IERC20 token);
```

Recommended Mitigation Remove them or use them where needed.