VaultGuardian Audit Report

Version 1.0

AGMASO Security Reviews

November 14, 2024

VaultGuardian Audit Report

AGMASO Security Reviews

14 nov 2024

Prepared by: AGMASO Security Reviews Lead Auditors:

- Alejandro G.

Table of Contents

Disclaimer

The AGMASO Security Reviews 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

The findings described in this document correspond the following commit hash:

```
XXXX
```

Scope

```
./src/
#-- abstract
|   #-- AStaticTokenData.sol
|   #-- AStaticUSDCData.sol
|   #-- AStaticWethData.sol
#-- dao
|   #-- VaultGuardianGovernor.sol
|   #-- VaultGuardianToken.sol
```

```
#-- interfaces
|   #-- IVaultData.sol
|   #-- IVaultGuardians.sol
|   #-- IVaultShares.sol
|   #-- InvestableUniverseAdapter.sol
#-- protocol
|   #-- VaultGuardians.sol
|   #-- VaultGuardiansBase.sol
|   #-- VaultShares.sol
|   #-- investableUniverseAdapters
|       #-- AaveAdapter.sol
|       #-- UniswapAdapter.sol
#-- vendor
    #-- DataTypes.sol
    #-- IPool.sol
    #-- IUniswapV2Factory.sol
    #-- IUniswapV2Router01.sol
```

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.

Roles

There are 4 main roles associated with the system.

- *Vault Guardian DAO*: The org that takes a cut of all profits, controlled by the `VaultGuardianToken`. The DAO that controls a few variables of the protocol, including:
  - `s_guardianStakePrice`
  - `s_guardianAndDaoCut`
  - And takes a cut of the ERC20s made from the protocol
- *DAO Participants*: Holders of the `VaultGuardianToken` who vote and take profits on the protocol
- *Vault Guardians*: Strategists/hedge fund managers who have the ability to move assets in and out of the investable universe. They take a cut of revenue from the protocol.
- *Investors*: The users of the protocol. They deposit assets to gain yield from the investments of the Vault Guardians.

Executive Summary

The Vault Guardians project takes novel approaches to work ERC-4626 into a hedge fund of sorts, but makes some large mistakes on tracking balances and profits.

Issues Found

Severity	Number of issues found
High	5
Medium	2
Low	3
Info	16
Gas	0
Total	26

High

[H-1] MEV FrontRunning attack posible becuase of lack of UniswapV2 slippage protection in `UniswapAdapter::_uniswapInvest` and `UniswapAdapter::_uniswapDivest`.

Description: In `UniswapAdapter::_uniswapInvest` the protocol swaps half of an ERC20 token so that they can invest in both sides of a Uniswap pool. It calls the `swapExactTokensForTokens` function of the `UnisapV2Router01` contract , which has two input parameters to note:

```
    function swapExactTokensForTokens(
        uint256 amountIn,
@>      uint256 amountOutMin,
        address[] calldata path,
        address to,
@>      uint256 deadline
    )
```

Also this vulnerability is finded in the addLiquidityBlock inside the same function:

```

(
            uint256 tokenAmount,
            uint256 counterPartyTokenAmount,
```

```
            uint256 liquidity
        ) = i_uniswapRouter.addLiquidity({
                tokenA: address(token),
                tokenB: address(counterPartyToken),
                amountADesired: amountOfTokenToSwap + amounts[0],
                //amountADesired: amounts[0],
                amountBDesired: amounts[1],
@>                amountAMin: 0,
@>                amountBMin: 0,
                to: address(this),
                deadline: block.timestamp
            });
```

`UniswapAdapter::_uniswapDivest`

```

function _uniswapDivest(
        IERC20 token,
        uint256 liquidityAmount
    ) internal returns (uint256 amountOfAssetReturned) {
        IERC20 counterPartyToken = token == i_weth ? i_tokenOne :
            i_weth;

        // @audit-High: MEV Sandwich attack, because no Slippage
            protection
        (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)];

        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];
    }
```

The parameter `amountOutMin` represents how much of the minimum number of tokens it expects

to return. The `deadline` parameter represents when the transaction should expire.

As seen below, the `UniswapAdapter::_uniswapInvest` function sets those parameters to `0` and `block.timestamp`:

```
    uint256[] memory amounts = i_uniswapRouter.swapExactTokensForTokens
        (
         amountOfTokenToSwap,
@>       0,
         s_pathArray,
         address(this),
@>       block.timestamp
    );
```

Impact: This results in either of the following happening:

- Anyone (e.g., a frontrunning bot) sees this transaction in the mempool, pulls a flashloan and swaps on Uniswap to tank the price before the swap happens, resulting in the protocol executing the swap at an unfavorable rate.
- Due to the lack of a deadline, the node who gets this transaction could hold the transaction until they are able to profit from the guaranteed swap.

Proof of Concept:

1. User calls `VaultShares::deposit` with a vault that has a Uniswap allocation.
    1. This calls `_uniswapInvest` for a user to invest into Uniswap, and calls the router's `swapExactTokensForTokens` function.
2. In the mempool, a malicious user could:
    1. Hold onto this transaction which makes the Uniswap swap
    2. Take a flashloan out
    3. Make a major swap on Uniswap, greatly changing the price of the assets
    4. Execute the transaction that was being held, giving the protocol as little funds back as possible due to the `amountOutMin` value set to 0.

This could potentially allow malicious MEV users and frontrunners to drain balances.

Recommended Mitigation:

For the deadline issue, we recommend the following:

DeFi is a large landscape. For protocols that have sensitive investing parameters, add a custom parameter to the `deposit` function so the Vault Guardians protocol can account for the customizations of DeFi projects that it integrates with.

In the `deposit` function, consider allowing for custom data.

```
- function deposit(uint256 assets, address receiver) public override(
    ERC4626, IERC4626) isActive returns (uint256) {
+ function deposit(uint256 assets, address receiver, bytes customData)
    public override(ERC4626, IERC4626) isActive returns (uint256) {
```

This way, you could add a `deadline` to the Uniswap swap, and also allow for more DeFi custom integrations.

For the `amountOutMin` issue, we recommend one of the following:

1. Do a price check on something like a Chainlink price feed before making the swap, reverting if the rate is too unfavorable.
2. Only deposit 1 side of a Uniswap pool for liquidity. Don't make the swap at all. If a pool doesn't exist or has too low liquidity for a pair of ERC20s, don't allow investment in that pool.

Note that these recommendation require significant changes to the codebase.

[H-2] `ERC4626::totalAssets` checks the balance of vault's underlying asset even when the asset is invested, resulting in incorrect values being returned

Description: The `ERC4626::totalAssets` function checks the balance of the underlying asset for the vault using the `balanceOf` function.

```
function totalAssets() public view virtual returns (uint256) {
    return _asset.balanceOf(address(this));
}
```

However, the assets are invested in the investable universe (Aave and Uniswap) which means this will never return the correct value of assets in the vault.

Impact: This breaks many functions of the `ERC4626` contract:

- `totalAssets`
- `convertToShares`
- `convertToAssets`
- `previewWithdraw`
- `withdraw`
- `deposit`

All calculations that depend on the number of assets in the protocol would be flawed, severely disrupting the protocol functionality.

Proof of Concept:

Code

```
1
2  function test_auditDepositByUserGivingToMuchShares() public hasGuardian
       {
3          console.log("Total supply de shares: ", wethVaultShares.
               totalSupply());
4          console.log("Total Weth de protocolo: ", wethVaultShares.
               totalAssets());
5
6          uint256 amountTodeposit = 10 ether;
7          weth.mint(amountTodeposit, user);
8          vm.startPrank(user);
9          weth.approve(address(wethVaultShares), amountTodeposit);
10
11         console.log("Total Weth de protocolo: ", wethVaultShares.
               totalAssets());
12         // Total Weth is used inside "deposit()" to calculate shares.
               This is taking only the half of the
13         // assets in the procotol as is not counting the assets staked
               in uniswap and aave.
14         wethVaultShares.deposit(amountTodeposit, user);
15         vm.stopPrank();
16
17         //Shares should be less than 10e18 shares. But it fails because
                is logging
18         // VaultShares__esto es shares previewDeposit:
               2003999999999999997
19
20         assert(wethVaultShares.balanceOf(user) <= 10 ether);
21     }
```

Recommended Mitigation: Do not use the OpenZeppelin implementation of the ERC4626 contract. Instead, natively keep track of users total amounts sent to each protocol. Potentially have an automation tool or some incentivised mechanism to keep track of protocol's profits and losses, and take snapshots of the investable universe.

This would take a considerable re-write of the protocol.

[H-3] Guardians can infinitely mint `VaultGuardianTokens` and take over DAO, stealing DAO fees and maliciously setting parameters

Description: Becoming a guardian comes with the perk of getting minted Vault Guardian Tokens (vgTokens). Whenever a guardian successfully calls `VaultGuardiansBase::becomeGuardian` or `VaultGuardiansBase::becomeTokenGuardian`, `_becomeTokenGuardian` is executed, which mints the caller `i_vgToken`.

```
    function _becomeTokenGuardian(IERC20 token, VaultShares tokenVault)
        private returns (address) {
        s_guardians[msg.sender][token] = IVaultShares(address(
            tokenVault));
@>      i_vgToken.mint(msg.sender, s_guardianStakePrice);
        emit GuardianAdded(msg.sender, token);
        token.safeTransferFrom(msg.sender, address(this),
            s_guardianStakePrice);
        token.approve(address(tokenVault), s_guardianStakePrice);
        tokenVault.deposit(s_guardianStakePrice, msg.sender);
        return address(tokenVault);
    }
```

Guardians are also free to quit their role at any time, calling the `VaultGuardianBase::quitGuardian` function. The combination of minting vgTokens, and freely being able to quit, results in users being able to farm vgTokens at any time.

Impact: Assuming the token has no monetary value, the malicious guardian could accumulate tokens until they can overtake the DAO. Then, they could execute any of these functions of the `VaultGuardians` contract:

```
  "sweepErc20s(address)": "942d0ff9",
  "transferOwnership(address)": "f2fde38b",
  "updateGuardianAndDaoCut(uint256)": "9e8f72a4",
  "updateGuardianStakePrice(uint256)": "d16fe105",
```

Proof of Concept:

1. User becomes WETH guardian and is minted vgTokens.
2. User quits, is given back original WETH allocation.
3. User becomes WETH guardian with the same initial allocation.
4. Repeat to keep minting vgTokens indefinitely.

Code

Place the following code into `VaultGuardiansBaseTest.t.sol`

```
    function testDaoTakeover() public hasGuardian hasTokenGuardian {
        address maliciousGuardian = makeAddr("maliciousGuardian");
        uint256 startingVoterUsdcBalance = usdc.balanceOf(
            maliciousGuardian);
        uint256 startingVoterWethBalance = weth.balanceOf(
            maliciousGuardian);
        assertEq(startingVoterUsdcBalance, 0);
        assertEq(startingVoterWethBalance, 0);

        VaultGuardianGovernor governor = VaultGuardianGovernor(payable(
            vaultGuardians.owner()));
```

```
9           VaultGuardianToken vgToken = VaultGuardianToken(address(
                governor.token()));
10
11          // Flash loan the tokens, or just buy a bunch for 1 block
12          weth.mint(mintAmount, maliciousGuardian); // The same amount as
                 the other guardians
13          uint256 startingMaliciousVGTokenBalance = vgToken.balanceOf(
                maliciousGuardian);
14          uint256 startingRegularVGTokenBalance = vgToken.balanceOf(
                guardian);
15          console.log("Malicious vgToken Balance:\t",
                startingMaliciousVGTokenBalance);
16          console.log("Regular vgToken Balance:\t",
                startingRegularVGTokenBalance);
17
18          // Malicious Guardian farms tokens
19          vm.startPrank(maliciousGuardian);
20          weth.approve(address(vaultGuardians), type(uint256).max);
21          for (uint256 i; i < 10; i++) {
22              address maliciousWethSharesVault = vaultGuardians.
                    becomeGuardian(allocationData);
23              IERC20(maliciousWethSharesVault).approve(
24                  address(vaultGuardians),
25                  IERC20(maliciousWethSharesVault).balanceOf(
                        maliciousGuardian)
26              );
27              vaultGuardians.quitGuardian();
28          }
29          vm.stopPrank();
30
31          uint256 endingMaliciousVGTokenBalance = vgToken.balanceOf(
                maliciousGuardian);
32          uint256 endingRegularVGTokenBalance = vgToken.balanceOf(
                guardian);
33          console.log("Malicious vgToken Balance:\t",
                endingMaliciousVGTokenBalance);
34          console.log("Regular vgToken Balance:\t",
                endingRegularVGTokenBalance);
35      }
```

Recommended Mitigation: There are a few options to fix this issue:

1. Mint vgTokens on a vesting schedule after a user becomes a guardian.
2. Burn vgTokens when a guardian quits.
3. Simply don't allocate vgTokens to guardians. Instead, mint the total supply on contract deployment.

[H-4] Missleading calculating the amount of Funds to redeem when the `_quitGuardian` function is called, leading to receive less funds than expected.

Description:

The function `_quitGuardian` is used to quit the guardian role. That Fn has to divest all the funds invested by the guardian in the Uniswap and Aave protocols and send the funds from the guardian to the guardian address. Inside this Function we have `maxRedeem` :

```
1
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          // @audit-high: maxRedeemable is calculated from the erc4626
               before the deinvest is executed,
8          // this means that the Max redeemable will be equal to the HOLD
                allocation.
9          uint256 maxRedeemable = tokenVault.maxRedeem(msg.sender);
10         uint256 numberOfAssetsReturned = tokenVault.redeem(
11             maxRedeemable,
12             msg.sender,
13             msg.sender
14         );
15         return numberOfAssetsReturned;
16     }
```

`maxRedeemable` is calculated from the ERC4626 before the deinvest is executed,this means that the Max redeemable amount will be equal to the HOLD allocation, not accounting the funds staked in the Uniswap and Aave protocols. This lead to a missfunction in the redeeming operation, and as result the Guardian will get less funds returned. The rest of those funds will be then stuck inside of the VaultSahres SC.

Impact:

The Guardian quitting the GuardianRole of the VaultShares will loose money as the Function will return only the amount allocated to HOLD. The rest of the deposited funds will be locked in the Vault, and not returned to the Gaurdian.

Proof of Concept:

PoC

```
1
2  function test_auditQuitGuardianOnlyreturnAmountFromHoldAllocation()
3          public
```

```
4           hasGuardian
5       {
6           console.log("Balance of Weth", weth.balanceOf(guardian));
7           console.log("Balance of Shares", wethVaultShares.balanceOf(
                guardian));
8           vm.startPrank(guardian);
9           wethVaultShares.approve(address(vaultGuardians), mintAmount);
10          uint256 returnedVaule = vaultGuardians.quitGuardian();
11          vm.stopPrank();
12          console.log("Returned Value", returnedVaule);
13          console.log("Balance of Weth after", weth.balanceOf(guardian));
14          console.log(
15              "Balance of Shares after",
16              wethVaultShares.balanceOf(guardian)
17          );
18          console.log(
19              "Balance of Weth Vaultsahre after",
20              weth.balanceOf(address(wethVaultShares))
21          );
22          //Demonstrate that maxReddemable is only refering to the HOLD
                allocated amount.
23          assert(
24              weth.balanceOf(address(wethVaultShares)) +
25                  weth.balanceOf(guardian) +
26                  vaultGuardians.getGuardianStakePrice() /
27                  2 ==
28                  mintAmount
29          );
30      }
```

Recommended Mitigation:

To mitigate this issue, dont use the ERC4626 function `tokenVault.maxRedeem(msg.sender)` before calling redeem funds. You need to find a new logic that tracks the total funds added by each depositor or guardian.

The point of this , is to track all the funds incluying the allocated funds in Hold, Uniswap and Aave protocols.

You could create a map to solve this issue. Note that those changes implicate a deep modification of the actual codebase.

[H-5] `UniswapAdapter::_uniswapInvest` breaks the Allocation setup, leading to not respect the allocation amounts setted before by the guardian.

Description:

We use `UniswapAdapter::_uniswapInvest` to invest in the Uniswap protocol. We need to stake

the same amount of Weth and its pair to the pool, for this reason we are using the next function to swap Weth and get the other token. `amountOfTokenToSwap` will be the half of the total Allocated funds for Uniswap.

```
1
2           uint256[] memory amounts = i_uniswapRouter.
                swapExactTokensForTokens({
3               amountIn: amountOfTokenToSwap,
4               amountOutMin: 0, // amountOutMin: amountOfTokenToSwap
5               path: s_pathArray,
6               to: address(this),
7               deadline: block.timestamp
8           });
```

Thus function returns an array with the `amounts[0]` for the weth amount swapped and `amounts[1]` for the Usdc amount swapped.

The issue is that the protocol is apporving and giving as a parameter for the function `addLiquidity()` the sum of `amountOfTokenToSwap + amounts[0]` which is double of what we wish to stake in the protocol as a `amountADesired`, resulting in a wrong amount of weth staked in the protocol and exceeding the Amount Allocations designed for Uniswap. Other impact is that the amount of HOLD Allocation weth in the vault will be less than it was desgined and setted by the Guardian.

```
1           succ = token.approve(
2               address(i_uniswapRouter),
3               amountOfTokenToSwap + amounts[0]
4               //amounts[0]
5           );
6
7           // @audit-Info: wrong comments, amounts[1] should be the
                counterPartyToken we got back if the token is equal to Weth
8           // and reverse in the other situation.
9           // amounts[1] should be the WETH amount we got back
10
11          // @audit-High: breaks AllocationData. You have to set
                amountADesired with the amounts[0], which is exactly the
12          // amount that has been used for the swap
13          // @audit-High: No slippage protection. SandWich attack
                possible.
14
15          (
16              uint256 tokenAmount,
17              uint256 counterPartyTokenAmount,
18              uint256 liquidity
19          ) = i_uniswapRouter.addLiquidity({
20                  tokenA: address(token),
21                  tokenB: address(counterPartyToken),
22                  amountADesired: amountOfTokenToSwap + amounts[0],
23                  //amountADesired: amounts[0],
```

```
                amountBDesired: amounts[1],
                amountAMin: 0,
                amountBMin: 0,
                to: address(this),
                deadline: block.timestamp
            });
```

Impact: High likeliihood, it will allways happen. High imapact as it will break the ALlocation invariant, and mess up the enterire protocol.

Proof of Concept:

PoC

```

function test_auditBrokenAllocationDataHold() public hasGuardian {
        console.log(
            "Esto es el balance de weth de VaultShare: ",
            weth.balanceOf(address(wethVaultShares))
        );

        uint256 holdAllocationWeth = vaultGuardians.
            getGuardianStakePrice() / 2;
        console.log(holdAllocationWeth);

        //Expected to fail
        assert(holdAllocationWeth == weth.balanceOf(address(
            wethVaultShares)));
    }
```

Recommended Mitigation:

To mitigate this issue you need to change few lines:

```

    succ = token.approve(
             address(i_uniswapRouter),
-             amountOfTokenToSwap + amounts[0]
+             amounts[0]
        );
        if (!succ) {
            revert UniswapAdapter__TransferFailed();
        }
        // @audit-Info: wrong comments, amounts[1] should be the
            counterPartyToken we got back if the token is equal to Weth
        // and reverse in the other situation.
        // amounts[1] should be the WETH amount we got back

        // @audit-High: breaks AllocationData. You have to set
            amountADesired with the amounts[0], which is exactly the
        // amount that has been used for the swap
```

```
16          // @audit-High: No slippage protection. SandWich attack
                possible.
17
18          (
19              uint256 tokenAmount,
20              uint256 counterPartyTokenAmount,
21              uint256 liquidity
22          ) = i_uniswapRouter.addLiquidity({
23                  tokenA: address(token),
24                  tokenB: address(counterPartyToken),
25  -               amountADesired: amountOfTokenToSwap + amounts[0],
26  +               amountADesired: amounts[0],
27                  amountBDesired: amounts[1],
28                  amountAMin: 0,
29                  amountBMin: 0,
30                  to: address(this),
31                  deadline: block.timestamp
32              });
```

Medium

[M-1] Potentially incorrect voting period and delay in governor may affect governance

The `VaultGuardianGovernor` contract, based on OpenZeppelin Contract's Governor, implements two functions to define the voting delay (`votingDelay`) and period (`votingPeriod`). The contract intends to define a voting delay of 1 day, and a voting period of 7 days. It does it by returning the value `1 days` from `votingDelay` and `7 days` from `votingPeriod`. In Solidity these values are translated to number of seconds.

However, the `votingPeriod` and `votingDelay` functions, by default, are expected to return number of blocks. Not the number seconds. This means that the voting period and delay will be far off what the developers intended, which could potentially affect the intended governance mechanics.

Consider updating the functions as follows:

```
1  function votingDelay() public pure override returns (uint256) {
2  -   return 1 days;
3  +   return 7200; // 1 day
4  }
5
6  function votingPeriod() public pure override returns (uint256) {
7  -   return 7 days;
8  +   return 50400; // 1 week
9  }
```

[M-2] Malicious User could gain 100% of voting power, and become owner of VaultGuardians.

Description:

When the protocol starts, the first guardian has teh oportunity to gain control of the DAO. When this first malicious user becomes a Vault Guardian, he will receive all the vgToken DAO tokens in the system till that moment. Meaning that he has the 100% of the vote power. This allows him to propose a Proposal and be voted by him getting this proposal to be executed by the Governor contract.

For example he could gain the ownership of the VaultGuardians smartContract sending as proposal a calldata calling `transferOwnership` of the same contract(becuase is Ownable).

Being the owner of VaultGuardians open a window of vulnerabilities that the attacker can executed, for instance, he can steal money of the DAO protocol using the function `sweepErc20s` to get all the rest tokens remaining in that contract.

Impact:

Impact will be high, as a malicious actor can gain control of many functionalities of the protocol, even managing to steal money. Likelihood is Medium. This will be possible only once, at the beginnig of the protocol. If someone else gets to be a Guardian, this person will get also VgTokens so the attacker will not have anymore all the voting power.

Proof of Concept:

PoC

```
1
2  function
       test_auditSendProposalToGainOwnershipOfVaultGuardiansAndDrainFunds()
3          public
4          hasGuardian
5      {
6          console.log(vaultGuardianToken.totalSupply());
7          bytes memory callData = abi.encodeWithSignature(
8              "transferOwnership(address)",
9              address(guardian)
10         );
11
12         address[] memory targets = new address[](1);
13         targets[0] = address(vaultGuardians);
14
15         uint256[] memory values = new uint256[](1);
16         values[0] = 0;
17
18         bytes[] memory calldatas = new bytes[](1);
19         calldatas[0] = callData;
20
21         vm.startPrank(guardian);
```

```
22          vaultGuardianToken.delegate(guardian);
23          uint256 proposalId = vaultGuardianGovernor.propose(
24              targets,
25              values,
26              calldatas,
27              "gain ownership of VaultGuardian SC"
28          );
29          vm.stopPrank();
30          console.log(proposalId);
31          vm.warp(block.timestamp + vaultGuardianGovernor.votingDelay() +
                1);
32          vm.roll(86402);
33          console.log(block.number);
34
35          //console.log("Esto es proposaleState", uint256(proposalState))
                ;
36          //cast a vote
37          vm.startPrank(guardian);
38          vaultGuardianGovernor.castVote(proposalId, 1);
39
40          vm.stopPrank();
41
42          vm.warp(block.timestamp + vaultGuardianGovernor.votingPeriod()
                + 1);
43          vm.roll(691202);
44          IGovernor.ProposalState proposalState = vaultGuardianGovernor.
                state(
45              proposalId
46          );
47          console.log("Esto es proposaleState", uint256(proposalState));
48
49          if (proposalState == IGovernor.ProposalState.Succeeded) {
50              vaultGuardianGovernor.execute(
51                  targets,
52                  values,
53                  calldatas,
54                  keccak256(bytes("gain ownership of VaultGuardian SC"))
55              );
56          }
57          proposalState = vaultGuardianGovernor.state(proposalId);
58          console.log("Esto es proposaleState", uint256(proposalState));
59
60          assert(vaultGuardians.owner() == address(guardian));
61
62          // We can now do everything possible for example call
                sweepErc20s() and get the money from the vaultsGuradians
63          // or change the dao&guardianCut.
64
65      }
```

Recommended Mitigation:

We recommend you to mint the totalsupply of VgTokens at the deployment time. After this, you distribute as you want the VgTokens already created to the Users. With this approach you avoid that the first Guardian can have all the voting power, in the case no one else wanst to be also user.

Low

[L-1] Incorrect vault name and symbol

When new vaults are deployed in the `VaultGuardianBase::becomeTokenGuardian` function, symbol and vault name are set incorrectly when the `token` is equal to `i_tokenTwo`. Consider modifying the function as follows, to avoid errors in off-chain clients reading these values to identify vaults.

```
else if (address(token) == address(i_tokenTwo)) {
    tokenVault =
    new VaultShares(IVaultShares.ConstructorData({
        asset: token,
-       vaultName: TOKEN_ONE_VAULT_NAME,
+       vaultName: TOKEN_TWO_VAULT_NAME,
-       vaultSymbol: TOKEN_ONE_VAULT_SYMBOL,
+       vaultSymbol: TOKEN_TWO_VAULT_SYMBOL,
        guardian: msg.sender,
        allocationData: allocationData,
        aavePool: i_aavePool,
        uniswapRouter: i_uniswapV2Router,
        guardianAndDaoCut: s_guardianAndDaoCut,
        vaultGuardian: address(this),
        weth: address(i_weth),
        usdc: address(i_tokenOne)
    }));
```

Also, add a new test in the `VaultGuardiansBaseTest.t.sol` file to avoid reintroducing this error, similar to what's done in the test `testBecomeTokenGuardianTokenOneName`.

[L-2] Unassigned return value when divesting AAVE funds

[L-3] : The nonReentrant `modifier` should occur before all other modifiers

This is a best-practice to protect against reentrancy in other modifiers.

4 Found Instances

- Found in src/protocol/VaultShares.sol Line: 181

```
        nonReentrant
```

- Found in src/protocol/VaultShares.sol Line: 242

```
    function rebalanceFunds() public isActive divestThenInvest
        nonReentrant {}
```

- Found in src/protocol/VaultShares.sol Line: 266

```
        nonReentrant
```

- Found in src/protocol/VaultShares.sol Line: 289

```
        nonReentrant
```

The `AaveAdapter::_aaveDivest` function is intended to return the amount of assets returned by AAVE after calling its `withdraw` function. However, the code never assigns a value to the named return variable `amountOfAssetReturned`. As a result, it will always return zero.

While this return value is not being used anywhere in the code, it may cause problems in future changes. Therefore, update the `_aaveDivest` function as follows:

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

[I-1] Dead Code Interface not used anywhere, causing less readability.

Description: `IInvestableUniverseAdapter` is not used anywhere, delete for mor clarity

```

// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;

import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

// @audit: dead interface
interface IInvestableUniverseAdapter {
    // function invest(IERC20 token, uint256 amount) external;
    // function divest(IERC20 token, uint256 amount) external;
}
```

also found here:

```

// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;

// @audit: DeadCode, Why is here nothing?
interface IVaultGuardians {}
```

Recommended Mitigation:

Delete this code if you will not use anywhere and make the codebase more easy to understand.

[I-2] Discrepancy between Documentation and NatSpec for `becomeGuardian` Function, creating confussion for the developer/auditor.

Description: There is a discrepancy between the documentation and the NatSpec comments for the `becomeGuardian` function. The documentation does not mention any payment of a fee in native ETH to execute `becomeGuardian`, whereas the NatSpec indicates the opposite. Furthermore, the code does not reflect any such ETH payment, and it would be impossible to implement this as the function is not marked as payable.

```

/*
     * @notice allows a user to become a guardian

@>     * @notice they have to send an ETH amount equal to the fee, and a WETH amount equal to the stake    price
     *
     * @param wethAllocationData the allocation data for the WETH vault
     */


    function becomeGuardian(
        AllocationData memory wethAllocationData
    ) external returns (address) {
        // creacion nuevo VaultShare
        VaultShares wethVault = new VaultShares(
            IVaultShares.ConstructorData({
                asset: i_weth, //i_weth viene de AStaticTokenData
                vaultName: WETH_VAULT_NAME,
                vaultSymbol: WETH_VAULT_SYMBOL,
                guardian: msg.sender,
                allocationData: wethAllocationData,
                aavePool: i_aavePool,
                uniswapRouter: i_uniswapV2Router,
                guardianAndDaoCut: s_guardianAndDaoCut,
                // q esta esto correcto?
```

```
                vaultGuardians: address(this),
                weth: address(i_weth),
                usdc: address(i_tokenOne)
            })
        );
        return _becomeTokenGuardian(i_weth, wethVault);
    }
```

Recommended Mitigation:

Delete the wrong natspec or redesign your function to be payable and add checks to require the payment of this Ethereum fee.

[I-3] Same name for Functions with different context and usage of same error for both functions with the same name, creates confussion and more difficult to read the codebase.

Description: You are using same name for functions with different context.

```

function quitGuardian() external onlyGuardian(i_weth) returns (uint256) {
        if (_guardianHasNonWethVaults(msg.sender)) {
            revert VaultGuardiansBase__CantQuitWethWithThisFunction();
        }
        return _quitGuardian(i_weth);
    }

function quitGuardian(
        IERC20 token
    ) external onlyGuardian(token) returns (uint256) {
        if (token == i_weth) {
            revert VaultGuardiansBase__CantQuitWethWithThisFunction();
        }
        return _quitGuardian(token);
    }
```

VaultGuardiansBase SC

```

  function updateHoldingAllocation(
        IERC20 token,
        AllocationData memory tokenAllocationData
    ) external onlyGuardian(token) {
        emit GuardianUpdatedHoldingAllocation(msg.sender, token);
        s_guardians[msg.sender][token].updateHoldingAllocation(
            tokenAllocationData
        );
    }
```

VaultShares SC

```
1
2  /**
3       * @notice Allows Vault Guardians to update their allocation ratio
           (and thus, their strategy of investment)
4       * @param tokenAllocationData The new allocation data
5       */
6      function updateHoldingAllocation(
7          AllocationData memory tokenAllocationData
8      ) public onlyVaultGuardians isActive {
9          uint256 totalAllocation = tokenAllocationData.holdAllocation +
10             tokenAllocationData.uniswapAllocation +
11             tokenAllocationData.aaveAllocation;
12         if (totalAllocation != ALLOCATION_PRECISION) {
13             revert VaultShares__AllocationNot100Percent(totalAllocation
                   );
14         }
15         s_allocationData = tokenAllocationData;
16         emit UpdatedAllocation(tokenAllocationData);
17     }
```

Recommended Mitigation:

1. Differenciate more the naming of the functions so we don't get confused.
2. Create diferent Error naming. This could lead to confussion.

[I-4] Poor clarity in the code sintaxis, leads to dificult to read.

Description:

```
1
2   function _guardianHasNonWethVaults(
3           address guardian
4       ) private view returns (bool) {
5   @>          if (address(s_guardians[guardian][i_tokenOne]) != address(0))
        {
6               return true;
7           } else {
8   @>              return address(s_guardians[guardian][i_tokenTwo]) !=
        address(0);
9           }
10      }
```

Recommended Mitigation:

Do the same sintaxis in both blocks.

[I-5] Mock not simulating the logic of the Lending protocol, causing issues to simulate real behavior inside of the unit tests.

Description: Mock not simulating the logic of the Lending protocol. Only transfering tokens and no generating awethTokenMock to generate interest.

```
1
2  function supply(address asset, uint256 amount, address, /* onBehalfOf
       */ uint16 /* referralCode */ ) external {
3          IERC20(asset).transferFrom(msg.sender, address(this), amount);
4      }
```

[I-6] Mock not generating LpTOkens, not simulating the logic of the Real pool.

Description: Mock not simulating the logic of the Lending protocol. Only transfering tokens and no generating awethTokenMock to generate interest.

```
1
2  function addLiquidity(
3          address tokenA,
4          address tokenB,
5          uint256 amountADesired,
6          uint256 amountBDesired,
7          uint256,
8          uint256,
9          address,
10         uint256
11     ) external returns (uint256 amountA, uint256 amountB, uint256
           liquidity) {
12         ERC20Mock(tokenA).transferFrom(msg.sender, address(this),
               amountADesired);
13         ERC20Mock(tokenB).transferFrom(msg.sender, address(this),
               amountBDesired);
14         _mint(msg.sender, liquidity);
15         return (amountADesired, amountBDesired, 0);
16     }
```

[I-7] Mock not burning LpTOkens, not simulating the logic of the Real pool

Description: Mock not burning LpTOkens, not simulating the logic of the Real pool

```
1  function removeLiquidity(address tokenA, address tokenB, uint256,
       uint256, uint256, address to, uint256)
2          external
3          returns (uint256 amountA, uint256 amountB)
4      {
```

```
5           uint256 tokenABalance = ERC20Mock(tokenA).balanceOf(address(
                this));
6           uint256 tokenBBalance = ERC20Mock(tokenB).balanceOf(address(
                this));
7
8           ERC20Mock(tokenA).transfer(to, tokenABalance);
9           ERC20Mock(tokenB).transfer(to, tokenBBalance);
10          return (tokenABalance, tokenBBalance);
11      }
```

[I-8]: Centralization Risk for trusted owners

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

5 Found Instances

- Found in src/dao/VaultGuardianToken.sol Line: 12

```
1  contract VaultGuardianToken is ERC20, ERC20Permit, ERC20Votes,
       Ownable {
```

- Found in src/dao/VaultGuardianToken.sol Line: 37

```
1      function mint(address to, uint256 amount) external onlyOwner {
```

- Found in src/protocol/VaultGuardians.sol Line: 41

```
1  contract VaultGuardians is Ownable, VaultGuardiansBase {
```

- Found in src/protocol/VaultGuardians.sol Line: 84

```
1      ) external onlyOwner {
```

- Found in src/protocol/VaultGuardians.sol Line: 98

```
1      function updateGuardianAndDaoCut(uint256 newCut) external
           onlyOwner {
```

[I-9]: Unsafe ERC20 Operations should not be used

ERC20 functions may not behave as expected. For example: return values are not always meaningful. It is recommended to use OpenZeppelin's SafeERC20 library.

5 Found Instances

- Found in src/protocol/VaultGuardiansBase.sol Line: 347

```
1           bool succ = token.approve(address(tokenVault),
               s_guardianStakePrice);
```

- Found in src/protocol/investableUniverseAdapters/AaveAdapter.sol Line: 29

```
1           bool succ = asset.approve(address(i_aavePool), amount);
```

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 62

```
1           bool succ = token.approve(
```

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 84

```
1           succ = counterPartyToken.approve(address(i_uniswapRouter),
                amounts[1]);
```

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 92

```
1           succ = token.approve(
```

[I-10]: Missing checks for `address(0)` when assigning values to address state variables

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

1 Found Instances

- Found in src/protocol/VaultGuardiansBase.sol Line: 343

```
1           s_guardians[msg.sender][token] = IVaultShares(address(
               tokenVault));
```

[I-11]: `public` functions not used internally could be marked `external`

Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

9 Found Instances

- Found in src/dao/VaultGuardianGovernor.sol Line: 19

```
1       function votingDelay() public pure override returns (uint256)
           {
```

- Found in src/dao/VaultGuardianGovernor.sol Line: 23

```
1     function votingPeriod() public pure override returns (uint256)
          {
```

- Found in src/dao/VaultGuardianGovernor.sol Line: 29

```
1     function quorum(uint256 blockNumber)
```

- Found in src/dao/VaultGuardianToken.sol Line: 30

```
1     function nonces(
```

- Found in src/protocol/VaultShares.sol Line: 146

```
1     function setNotActive() public onlyVaultGuardians isActive {
```

- Found in src/protocol/VaultShares.sol Line: 174

```
1     function deposit(
```

- Found in src/protocol/VaultShares.sol Line: 242

```
1     function rebalanceFunds() public isActive divestThenInvest
          nonReentrant {}
```

- Found in src/protocol/VaultShares.sol Line: 258

```
1     function withdraw(
```

- Found in src/protocol/VaultShares.sol Line: 281

```
1     function redeem(
```

[I-12]: Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

12 Found Instances

- Found in src/protocol/VaultGuardians.sol Line: 49

```
1     event VaultGuardians__UpdatedStakePrice(
```

- Found in src/protocol/VaultGuardians.sol Line: 53

```
1       event VaultGuardians__UpdatedFee(uint256 oldFee, uint256
            newFee);
```

- Found in src/protocol/VaultGuardians.sol Line: 54

```
1       event VaultGuardians__SweptTokens(address asset);
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 94

```
1       event GuardianAdded(address guardianAddress, IERC20 token);
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 95

```
1       event GaurdianRemoved(address guardianAddress, IERC20 token);
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 96

```
1       event InvestedInGuardian(
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 101

```
1       event DinvestedFromGuardian(
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 106

```
1       event GuardianUpdatedHoldingAllocation(
```

- Found in src/protocol/VaultShares.sol Line: 45

```
1       event UpdatedAllocation(AllocationData allocationData);
```

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 19

```
1       event UniswapInvested(
```

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 24

```
1       event UniswapDivested(uint256 tokenAmount, uint256 wethAmount)
            ;
```

- Found in src/vendor/IUniswapV2Factory.sol Line: 9

```
1       event PairCreated(
```

[I-13]: PUSH0 is not supported by all chains

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

18 Found Instances

- Found in src/abstract/AStaticTokenData.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/abstract/AStaticUSDCData.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/abstract/AStaticWethData.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/dao/VaultGuardianGovernor.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/dao/VaultGuardianToken.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/interfaces/IVaultData.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/interfaces/IVaultGuardians.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/interfaces/IVaultShares.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/interfaces/InvestableUniverseAdapter.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/protocol/VaultGuardians.sol Line: 28

```
1 pragma solidity 0.8.20;
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 28

```
1 pragma solidity 0.8.20;
```

- Found in src/protocol/VaultShares.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/protocol/investableUniverseAdapters/AaveAdapter.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/vendor/DataTypes.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/vendor/IPool.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/vendor/IUniswapV2Factory.sol Line: 3

```
1 pragma solidity 0.8.20;
```

- Found in src/vendor/IUniswapV2Router01.sol Line: 2

```
1 pragma solidity 0.8.20;
```

[I-14]: Unused Custom Error

it is recommended that the definition be removed when custom error is unused

4 Found Instances

- Found in src/protocol/VaultGuardians.sol Line: 44

```
1     error VaultGuardians__TransferFailed();
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 53

```
1     error VaultGuardiansBase__NotEnoughWeth(
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 61

```
1     error VaultGuardiansBase__CantQuitGuardianWithNonWethVaults(
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 66

```
1     error VaultGuardiansBase__FeeTooSmall(uint256 fee, uint256
          requiredFee);
```

[I-15]: Potentially unused `private` / `internal` state variables found.

State variable appears to be unused. No analysis has been performed to see if any inilne assembly references it. So if that's not the case, consider removing this unused variable.

1 Found Instances

- Found in src/protocol/VaultGuardiansBase.sol Line: 80

```
1     uint256 private constant GUARDIAN_FEE = 0.1 ether;
```

[I-16]: Unused Imports

Redundant import statement. Consider removing it.

1 Found Instances

- Found in src/interfaces/InvestableUniverseAdapter.sol Line: 4

```
1 import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.
      sol";
```