Vault Guardians Protocol Audit Report

Version 1.0

GuireWire

July 16, 2024

Protocol Audit Report

GuireWire

July 16, 2024

Prepared by: GuireWire Lead Auditors: - GuireWire

Table of Contents

- Medium
  * [M-1] Potentially incorrect voting period and delay in governor may affect governance
- Low
  * [L-1] Incorrect vault name and symbol
  * [L-2] Unassigned return value when divesting AAVE funds
  * [L-3] Centralization Risk for trusted owners
  * [L-4] Unsafe ERC20 Operations should not be used
  * [L-5] Missing checks for `address(0)` when assigning values to address state variables
  * [L-6] **`public`** functions not used internally could be marked `external`
  * [L-7] Event is missing `indexed` fields
  * [L-8] The `nonReentrant modifier` should occur before all other modifiers
  * [L-9] PUSH0 is not supported by all chains
  * [L-10] Empty Block
  * [L-11] Unused Custom Error

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.

Disclaimer

The GuireWire 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

		Impact		
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

Commit Hash:

```
XXXXXXXXX
```

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
```

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.

Audit deemed to be successful as we identified 4 high severity and 1 medium severity vulnerabilities.

We spent 5 hours on the audit.

Issues found

Severity	Number of Issues Found
High	4
Medium	1
Low	11
Total	16

Findings

High

[H-1] Lack of UniswapV2 slippage protection in `UniswapAdapter::_uniswapInvest` enables frontrunners to steal profits

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
    )
```

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

Add the following code to the `VaultSharesTest.t.sol` file.

```
function testWrongBalance() public {
    // Mint 100 ETH
    weth.mint(mintAmount, guardian);
    vm.startPrank(guardian);
    weth.approve(address(vaultGuardians), mintAmount);
    address wethVault = vaultGuardians.becomeGuardian(allocationData);
    wethVaultShares = VaultShares(wethVault);
    vm.stopPrank();

    // prints 3.75 ETH
    console.log(wethVaultShares.totalAssets());

    // Mint another 100 ETH
    weth.mint(mintAmount, user);
    vm.startPrank(user);
    weth.approve(address(wethVaultShares), mintAmount);
    wethVaultShares.deposit(mintAmount, user);
    vm.stopPrank();

    // prints 41.25 ETH
    console.log(wethVaultShares.totalAssets());
}
```

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()));
        VaultGuardianToken vgToken = VaultGuardianToken(address(governor.token()));

        // Flash loan the tokens, or just buy a bunch for 1 block
        weth.mint(mintAmount, maliciousGuardian); // The same amount as the other guardians
        uint256 startingMaliciousVGTokenBalance = vgToken.balanceOf(maliciousGuardian);
        uint256 startingRegularVGTokenBalance = vgToken.balanceOf(guardian);
        console.log("Malicious vgToken Balance:\t", startingMaliciousVGTokenBalance);
        console.log("Regular vgToken Balance:\t", startingRegularVGTokenBalance);

        // Malicious Guardian farms tokens
        vm.startPrank(maliciousGuardian);
        weth.approve(address(vaultGuardians), type(uint256).max);
        for (uint256 i; i < 10; i++) {
            address maliciousWethSharesVault = vaultGuardians.becomeGuardian(allocationData);
            IERC20(maliciousWethSharesVault).approve(
                address(vaultGuardians),
                IERC20(maliciousWethSharesVault).balanceOf(maliciousGuardian)
            );
            vaultGuardians.quitGuardian();
        }
        vm.stopPrank();

        uint256 endingMaliciousVGTokenBalance = vgToken.balanceOf(maliciousGuardian);
        uint256 endingRegularVGTokenBalance = vgToken.balanceOf(
            guardian);
        console.log("Malicious vgToken Balance:\t",
            endingMaliciousVGTokenBalance);
        console.log("Regular vgToken Balance:\t",
            endingRegularVGTokenBalance);
    }
```

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] Using `block.timestamp` for swap deadline offers no protection

Description and Impact:

In the PoS model, proposers know well in advance if they will propose one or consecutive blocks ahead of time. In such a scenario, a malicious validator can hold back the transaction and execute it at a more favourable block number.Consider allowing function caller to specify swap deadline input parameter.

Proof of Concept:

2 Found Instances

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 54

  ```
          uint256[] memory amounts = i_uniswapRouter.
              swapExactTokensForTokens({
  ```

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 104

  ```
          uint256[] memory amounts = i_uniswapRouter.
              swapExactTokensForTokens({
  ```

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:

```
function votingDelay() public pure override returns (uint256) {
-    return 1 days;
+    return 7200; // 1 day
}

function votingPeriod() public pure override returns (uint256) {
-    return 7 days;
+    return 50400; // 1 week
}
```

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

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)
        });
}
```

[L-3] Centralization Risk for trusted owners

Description and Impact:

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

Proof of Concept:

5 Found Instances

- Found in src/dao/VaultGuardianToken.sol Line: 9

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

- Found in src/dao/VaultGuardianToken.sol Line: 21

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

- Found in src/protocol/VaultGuardians.sol Line: 40

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

- Found in src/protocol/VaultGuardians.sol Line: 71

```
    function updateGuardianStakePrice(uint256 newStakePrice)
        external onlyOwner {
```

- Found in src/protocol/VaultGuardians.sol Line: 82

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

[L-4] Unsafe ERC20 Operations should not be used

Description and Impact:

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

Proof of Concept:

5 Found Instances

- Found in src/protocol/VaultGuardiansBase.sol Line: 273

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

- Found in src/protocol/investableUniverseAdapters/AaveAdapter.sol Line: 25

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

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 50

```
        bool succ = token.approve(address(i_uniswapRouter),
            amountOfTokenToSwap);
```

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

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

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 66

```
            succ = token.approve(address(i_uniswapRouter),
                amountOfTokenToSwap + amounts[0]);
```

[L-5] Missing checks for `address(0)` when assigning values to address state variables

Description and Impact:

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

Proof of Concept:

1 Found Instances

- Found in src/protocol/VaultGuardiansBase.sol Line: 269

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

[L-6] `public` functions not used internally could be marked `external`

Description and Impact:

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

Proof of Concept:

9 Found Instances

- Found in src/dao/VaultGuardianGovernor.sol Line: 17

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

- Found in src/dao/VaultGuardianGovernor.sol Line: 21

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

- Found in src/dao/VaultGuardianGovernor.sol Line: 27

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

- Found in src/dao/VaultGuardianToken.sol Line: 17

```
        function nonces(address ownerOfNonce) public view override(
            ERC20Permit, Nonces) returns (uint256) {
```

- Found in src/protocol/VaultShares.sol Line: 115

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

- Found in src/protocol/VaultShares.sol Line: 140

```
    function deposit(uint256 assets, address receiver)
```

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

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

- Found in src/protocol/VaultShares.sol Line: 189

```
    function withdraw(uint256 assets, address receiver, address
       owner)
```

- Found in src/protocol/VaultShares.sol Line: 206

```
    function redeem(uint256 shares, address receiver, address
       owner)
```

[L-7] Event is missing `indexed` fields

Description and Impact:

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.

Proof of Concept:

12 Found Instances

- Found in src/protocol/VaultGuardians.sol Line: 48

```
    event VaultGuardians__UpdatedStakePrice(uint256 oldStakePrice,
        uint256 newStakePrice);
```

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

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

- Found in src/protocol/VaultGuardians.sol Line: 50

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

- Found in src/protocol/VaultGuardiansBase.sol Line: 78

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

- Found in src/protocol/VaultGuardiansBase.sol Line: 79

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

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

```
    event InvestedInGuardian(address guardianAddress, IERC20 token
        , uint256 amount);
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 81

```
    event DinvestedFromGuardian(address guardianAddress, IERC20
        token, uint256 amount);
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 82

```
    event GuardianUpdatedHoldingAllocation(address guardianAddress
        , IERC20 token);
```

- Found in src/protocol/VaultShares.sol Line: 35

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

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

```
    event UniswapInvested(uint256 tokenAmount, uint256 wethAmount,
         uint256 liquidity);
```

- Found in src/protocol/investableUniverseAdapters/UniswapAdapter.sol Line: 20

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

- Found in src/vendor/IUniswapV2Factory.sol Line: 7

```
    event PairCreated(address indexed token0, address indexed
        token1, address pair, uint256);
```

[L-8] The `nonReentrant` `modifier` should occur before all other modifiers

Description and Impact:

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

Proof of Concept:

4 Found Instances

- Found in src/protocol/VaultShares.sol Line: 144

```
            nonReentrant
```

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

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

- Found in src/protocol/VaultShares.sol Line: 193

```
            nonReentrant
```

- Found in src/protocol/VaultShares.sol Line: 210

```
            nonReentrant
```

[L-9] PUSH0 is not supported by all chains

Description and Impact:

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.

Proof of Concept:

18 Found Instances

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

  ```
  pragma solidity 0.8.20;
  ```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

[L-10] Empty Block

Description and Impact:

Consider removing empty blocks.

Proof of Concept:

1 Found Instances

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

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

[L-11] Unused Custom Error

Description and Impact:

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

Proof of Concept:

4 Found Instances

- Found in src/protocol/VaultGuardians.sol Line: 43

```
    error VaultGuardians__TransferFailed();
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 46

```
    error VaultGuardiansBase__NotEnoughWeth(uint256 amount,
        uint256 amountNeeded);
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 48

```
    error VaultGuardiansBase__CantQuitGuardianWithNonWethVaults(
       address guardianAddress);
```

- Found in src/protocol/VaultGuardiansBase.sol Line: 51

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