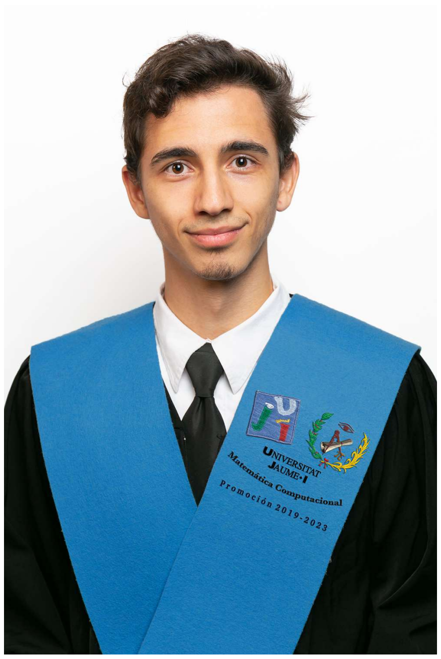
VaultGuardiansReport

CarlosAlbaWork

February 10, 2024



VaultGuardiansReport

Version 1.0

CarlosAlbaWork

February 10, 2024

Prepared by: CarlosAlbaWork Lead Auditors:

- Carlos Alba

Table of Contents

Disclaimer

CarlosAlbaWork 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

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	1
Medium	0
Low	6
Info	3
Gas	0
Total	10

[H-1] `VaultGuardiansBase::_quitGuardian` doesn´t burn the VgTokens minted in `VaultGuardiansBase::_becomeTokenGuardian` allowing guardians to get many VgTokens going in and out of the protocol

Description: The DAO gives some voting power to Guardians via VgTokens minted every time a guardian sets a pool. However, this voting power is never substracted once the guardian leaves the protocol, setting the perfect enviroment for Attackers to enter and exit as guardians, building power with every action
Impact: They would have the ability to accept any proposal they make with such quantity of voting tokens, putting in danger the protocol and its users funds
Proof of Concept:

```
function testQuitGuardianDoesntBurnVgtokens() public hasGuardian {
        vm.startPrank(guardian);
        wethVaultShares.approve(address(vaultGuardians), mintAmount);
        vaultGuardians.quitGuardian();
        vm.stopPrank();
        assertEq(vaultGuardianToken.balanceOf(guardian), 0);
    }
```

Recommended Mitigation:

Updating the `IVaultShares` interface

```
struct ConstructorData {
        IERC20 asset;
        string vaultName;
        string vaultSymbol;
        address guardian;
        AllocationData allocationData;
        address aavePool;
        address uniswapRouter;
+       uint256 guardianStakePrice;
        uint256 guardianAndDaoCut;
        address vaultGuardians;
        address weth;
        address usdc;
    }
    .
    .
    .
+   function getGuardianStakePrice() external view returns (uint256);
```

Adding the function `VaultGuardianToken::burn`

```
+function burn(address to, uint256 amount) external onlyOwner {
+        _burn(to, amount);
+    }
```

Adding the variable `VaultShares::i_guardianStakePrice` to the vault, updating the constructor and the getter

```
    uint256 private immutable i_guardianAndDaoCut;
+   uint256 private immutable i_guardianStakePrice;
    bool private s_isActive;
    .
    .
    .
    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_guardianStakePrice = constructorData.guardianStakePrice;
        i_guardianAndDaoCut = constructorData.guardianAndDaoCut;
    .
    .
    .
+    function getGuardianStakePrice() external view returns (uint256) {
+        return i_guardianStakePrice;
+    }
```

We finally can add a burn statement in `VaultGuardiansBase::quitGuardian` :

```
function _quitGuardian(IERC20 token) private returns (uint256) {
        IVaultShares tokenVault = IVaultShares(s_guardians[msg.sender][token]);
        s_guardians[msg.sender][token] = IVaultShares(address(0));
+       i_vgToken.burn(msg.sender, tokenVault.getGuardianStakePrice());
        emit GaurdianRemoved(msg.sender, token);
        tokenVault.setNotActive();
        uint256 maxRedeemable = tokenVault.maxRedeem(msg.sender);
        uint256 numberOfAssetsReturned = tokenVault.redeem(
            maxRedeemable,
            msg.sender,
            msg.sender
        );
        return numberOfAssetsReturned;
    }
```

We have to add the `VaultGuardiansBase::guardianStakePrice` of the time of the vault created because it can change in time in the main contract with change proposals, and thus not burning the amount of vgtoken minted when the guardian was added.

[L-1] `VaultGuardiansBase::becomeTokenGuardian` does not generate properly the vault when `token` is `i_tokenTwo`

Description: Using `VaultGuardiansBase::becomeTokenGuardian` you can become a guardian of a Vault. However, if the function is called with `token` being `i_tokenTwo`, the vault name and symbol are still set to the ones related to `i_tokenOne` **Impact:** Missinformation and confusion towards managing different vaults if the name and the token do not match **Proof of Concept:**

Recommended Mitigation:

```
tokenVault = new VaultShares(IVaultShares.ConstructorData({
                asset: token,
+                vaultName: TOKEN_TWO_VAULT_NAME,
+                vaultSymbol: TOKEN_TWO_VAULT_SYMBOL,
-                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)
            }));
```

[L-2] `VaultGuardiansBase::s_guardianStakePrice` can be set to 0 or a very low value with `VaultGuardians::updateGuardianStakePrice`, erasing the biggest incentive for guardians for acting correctly with other people's funds

Description: A proposal published by malicious attackers could set the `VaultGuardiansBase::s_guardianStakePrice` to a low number, disrupting the protocol functionality and destroying the main filter for people to become VaultGuardians

Impact:

Lowered Accountability: Without any stake requirement, there's less at risk for guardians in terms of their own capital. This could lead to less accountability and diligence in managing the vaults since guardians don't have a personal financial stake in their performance.

Increased Risk of Malicious Behavior: With no guardian stake requirement, there's a higher risk of malicious actors becoming guardians with the intention of mismanaging funds or engaging in fraudulent activities. They have less to lose personally if their actions negatively impact the vaults.

Decreased Confidence from Users: Users may be less confident in the reliability and security of the vault system if guardians aren't required to have a stake. They may perceive the system as less trustworthy and be hesitant to deposit their funds, leading to lower adoption and usage.

Weakened Incentives for Good Performance: The absence of a guardian stake reduces the incentive for guardians to perform well and maximize yields for users. Since there's no personal financial stake involved, guardians may be less motivated to actively manage the vaults and seek optimal investment strategies.

Challenges in System Governance: Without a guardian stake requirement, it may be more difficult to implement effective governance mechanisms within the system. Guardians without a stake may have less incentive to participate in governance processes or make decisions in the best interest of users.

Potential for System Instability: The lack of a guardian stake requirement could introduce instability into the system, as guardians may enter and exit the role more frequently without significant consequences. This could disrupt the continuity of vault management and lead to fluctuations in performance. **Proof of Concept:**

Recommended Mitigation:

Adding a constant that is the minimum stake that can be achieved and check in consequence. Then, checking the value is above that minimum, if not, throw a custom error. This could solve a problem in case the DAO gets hijacked by people with many voting power. Adding a Constant MINIMUM_STAKE_PRICE would set a barrier where Stake can´t go lower, making it harder for malicius voters to attack the protocol.

```
+ error VaultGuardians__NewStakeNotBigEnough() ;
+ uint256 private constant MINIMUM_STAKE_PRICE;
function updateGuardianStakePrice(uint256 newStakePrice) external onlyOwner {
+       if (newStakePrice < MINIMUM_STAKE_PRICE)  {
+           revert VaultGuardians__NewStakeNotBigEnough();
+       }
        s_guardianStakePrice = newStakePrice;
        emit VaultGuardians__UpdatedStakePrice(s_guardianStakePrice, newStakePrice);
    }
```

[L-3] `VaultGuardiansBase::s_guardianAndDaoCut` can be set to 100 (or even higher!) with `VaultGuardians::updateGuardianAndDaoCut`, erasing the biggest incentive for users to invest in new pools or breaking the logic of the contract

Description: A proposal published by malicious attackers could set the `VaultGuardiansBase::s_guardianAndDaoCut` to a high number, disrupting the protocol functionality and destroying the main reason for people to invest in new pools. If the number is more than 100% it can generate problems when setting up pools.

Impact: Misaligned Incentives: When vault guardians receive a disproportionately high share of the returns generated from investments, their incentives may become misaligned with the interests of the users who deposited their money in the vaults. Guardians may prioritize maximizing their own profits over acting in the best interests of the depositors, potentially leading to suboptimal investment decisions or even malfeasance.

Reduced Returns for Users: High percentage shares for guardians and DAOs mean less returns are passed on to the users who deposited their funds in the vaults. This can result in lower overall returns for users, diminishing the attractiveness of the vaults and potentially leading to dissatisfaction or withdrawal of funds.

Decreased Trust and Confidence: Users may lose trust and confidence in the vaults and the protocol as a whole if they perceive that guardians and DAOs are unfairly benefiting at their expense. This can erode trust in the platform, leading to decreased participation and adoption.

Limited Growth Potential: Excessive allocation of returns to guardians and DAOs may limit the growth potential of the protocol by disincentivizing users from depositing funds into the vaults. Without a sufficient user base and capital inflow, the protocol may struggle to achieve its intended objectives and expand its ecosystem. **Proof of Concept:**

Recommended Mitigation:

Adding a constant that is the maximum share ratio that can be achieved and check in consequence. Then, checking the value is below that maximum, if not, throw a custom error. This could solve a problem in case the DAO gets hijacked by people with many voting power. Adding a Constant MAXIMUM_SHARE_PRICE would set a barrier where Share can´t go higher, making it harder for malicius voters to attack the protocol.

```
+ error VaultGuardians__NewShareNotSmallEnough() ;
+ uint256 private constant MAXIMUM_SHARE_PRICE;
function updateGuardianAndDaoCut(uint256 newCut) external onlyOwner {
+       if (newCut > MAXIMUM_SHARE_PRICE)  {
+           revert VaultGuardians__NewShareNotSmallEnough();
+       }
        s_guardianAndDaoCut = newCut;
        emit VaultGuardians__UpdatedStakePrice(s_guardianAndDaoCut, newCut);
    }
```

[L-4] Event `VaultGuardians__UpdatedStakePrice` is used in `VaultGuardians::updateGuardianAndDaoCut` when `VaultGuardians__UpdatedFee` should be used instead

Description: The function `VaultGuardians::updateGuardianAndDaoCut` updates the fees the dao and guardians will get when managing users´ money in the vault. Therefore, the event emitted should be the one named `VaultGuardians__UpdatedFee`. However, it emits the event `VaultGuardians__UpdatedStakePrice` related to the `VaultGuardians::updateGuardianStakePrice` instead.

Impact: It gives Misleading information in the events, and can generate misunderstanding for third-party protocols and smart contracts listening to the events. **Proof of Concept:**

Recommended Mitigation: Implementing the suitable error(In the next code we take into account the changes made in the previous finding recommended mitigation):

```
error VaultGuardians__NewShareNotSmallEnough() ;
uint256 private constant MAXIMUM_SHARE_PRICE;
function updateGuardianAndDaoCut(uint256 newCut) external onlyOwner {
       if (newCut > MAXIMUM_SHARE_PRICE)  {
           revert VaultGuardians__NewShareNotSmallEnough();
       }
```

```
        s_guardianAndDaoCut = newCut;
-       emit VaultGuardians__UpdatedStakePrice(s_guardianAndDaoCut, newCut);
+       emit VaultGuardians__UpdatedFee(s_guardianAndDaoCut, newCut);
    }
```

[L-5] Functions `VaultGuardians::updateGuardianAndDaoCut` and `VaultGuardians::updateGuardianStakePrice` don´t emit in their respective events the oldValue and newvalue properly

Description: The function `VaultGuardians::updateGuardianAndDaoCut` and `VaultGuardians::updateGuardianStakePrice` should emit their respective events with the old and new value, but because `VaultGuardiansBase::s_guardianAndDaoCut` and `VaultGuardiansBase::s_guardianStakePrice` change in the middle of the function to the new value, this line emits twice the new value:`emit VaultGuardians__UpdatedFee(s_guardianAndDaoCut, newCut);`, `emit VaultGuardians__UpdatedStakePrice(s_guardianStakePrice, newStakePrice);`

Impact: It gives Misleading information in the events, and can generate misunderstanding for third-party protocols and smart contracts listening to the events.
Proof of Concept:

Recommended Mitigation: Same mitigation for both functions, adding an auxiliar variable containing the oldvalue (In the next code we take into account the changes made in the previous findings recommended mitigations):

```
error VaultGuardians__NewShareNotSmallEnough() ;
uint256 private constant MAXIMUM_SHARE_PRICE;
function updateGuardianAndDaoCut(uint256 newCut) external onlyOwner {
       if (newCut > MAXIMUM_SHARE_PRICE)  {
           revert VaultGuardians__NewShareNotSmallEnough();
       }
+       uint256 oldValue = s_guardianAndDaoCut;
        s_guardianAndDaoCut = newCut;
-       emit VaultGuardians__UpdatedFee(s_guardianAndDaoCut, newCut);
+       emit VaultGuardians__UpdatedFee(oldValue, newCut);
    }

...

error VaultGuardians__NewStakeNotBigEnough() ;
uint256 private constant MINIMUM_STAKE_PRICE;
function updateGuardianStakePrice(uint256 newStakePrice) external onlyOwner {
       if (newStakePrice < MINIMUM_STAKE_PRICE)  {
           revert VaultGuardians__NewStakeNotBigEnough();
       }
+       uint256 oldValue = s_guardianStakePrice;
        s_guardianStakePrice = newStakePrice;
-       emit VaultGuardians__UpdatedStakePrice(s_guardianStakePrice, newStakePrice);
+       emit VaultGuardians__UpdatedStakePrice(oldValue, newStakePrice);
    }
```

[L-6] The parameter returned in `AaveAdapter::_aaveDivest` is never used

Description: The parameter `amountOfAssetReturned` is never updated, so the function always returns 0 regardless of the value withdrawn.

Recommended Mitigation: Updating the value with the `withdraw` call result

```
-           i_aavePool.withdraw({
+ amountOfAssetReturned = i_aavePool.withdraw({
            asset: address(token),
            amount: amount,
            to: address(this)
        });
```

[I-1] `IInvestableUniverseAdapter` y `IVaultGuardians` are never used

Description: Those interfaces are never used or initialized. Maybe there is some extra codification forgotten to be implemented

[I-2] The modifier `nonReentrant` in `VaultShares` is after other modifiers, allowing reentrancies in those

Description: Even though no modifiers are reentrant in the current code, because this modifier is set after the other ones, doesn´t protect as expected because this other modifiers could have some reentracies that are not checked

Recommended Mitigation: The best practice is invoking the modifier first, in order to cover all the later code

[I-3] The events `VaultGuardiansBase::InvestedInGuardian` and `VaultGuardiansBase::DinvestedFromGuardian` are never used

Description: Those events are never used in any part of the code. Would be more gas efficient to delete them.