Vault Guardians Security Review

Version 1.0

palmcivet

April 3, 2024

Vault Guardians Security Review

palmcivet

April 3, 2024

Prepared by: Palmcivet

Table of Contents

    * [L-2] `VaultGuardians::updateGuardianAndDaoCut()` function incorrectly emits `VaultGuardians__UpdatedStakePrice` when it should be emitting `VaultGuardians__UpdatedFee`
    * [L-3] LINK Token Vault name and symbol are incorrectly assigned names meant for USDC vault in `VaultGuardiansBase::becomeTokenGuardian()` function
    * [L-4] Guardian Overwrites Existing Vault in `VaultGuardiansBase::s_guardians` Mapping
    * [L-5] Unused named return parameter in `AaveAdapter::_aaveDivest` always returns 0
    * [L-6] `UniswapAdapter::UniswapInvested` and `UniswapAdapter::UniswapDivested` events should return `counterpartyAmount` as the second parameter, instead of `wethAmount`

- Informational
    * [I-1] Natspec in `AStaticWethData` refers to "four tokens" when there are only 3
    * [I-2] Natspec in `VaultGuardians::updateGuardianAndDaoCut` describes calculation in reverse
    * [I-3] `VaultGuardians::sweepErc20s` lacks access control
    * [I-4] `VaultGuardiansBase::s_isApprovedToken` mapping is not used anywhere
    * [I-5] `VaultGuardiansBase::InvestedInGuardian` and `VaultGuardiansBase::DinvestedFromGuardian` events are not being used anywhere
    * [I-6] `VaultGuardiansBase::_becomeTokenGuardian` includes a lot of storage reads that can be reduced
    * [I-7] `VaultShares::onlyGuardian` modifier and `VaultShares::VaultShares__NotGua` error are not being used anywhere
    * [I-8] `VaultShares::deposit`, `VaultShares::withdraw` and `VaultShares::redeem` functions could be marked external
    * [I-9] `VaultGuardiansBase::GUARDIAN_FEE` constant isn't used
    * [I-10] Lack of zero address checks
    * [I-11] Lack of natspec for some functions
    * [I-12] `VaultGuardiansBase::becomeGuardian` natspec says user has to send ETH, but function is not payable
    * [I-13] `IInvestableUniverseAdapter` interface isn't used
    * [I-14] `IVaultGuardians` interface isn't used

Protocol Summary

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

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

Disclaimer

The palmcivet 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

- Commit Hash: 964de408365203e27b32d26ab6d8c4bfbcffa118

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

I spent a few weeks looking at the Vault Guardians codebase. I ignored the already known issues from the Cyfrin report and was glad to find another high severity issue. I learned a lot during this review.

Issues found

Severity	Number of issues found
High	1
Medium	0
Low	6
Informational	14
Total	21

Findings

High

[H-1] `VaultShares::deposit` mints additional shares to `i_guardian` and `i_vaultGuardians` without equivalent assets backing them, causing share dilution and value discrepancy

Description: The `deposit` function in the `VaultShares` contract mints shares for the guardian (`i_guardian`) and the vault guardians (`i_vaultGuardians`) as part of the deposit process, in addition to the shares minted for the depositor. This action is performed without adding an equivalent amount of underlying assets to the vault to back these additional shares. As a result, the total supply of shares increases without a corresponding increase in the vault's assets, leading to a dilution of the value of all shares.

Impact: This mechanism dilutes the value of each share by increasing the total number of shares without increasing the vault's total assets. Shareholders find their shares representing a smaller fraction of the vault's assets, meaning their holdings lose value. It undermines the principle of proportional ownership inherent to tokenized vaults and could deter future investments due to perceived unfairness in share distribution.

Proof of Concept:

1. Consider a scenario where assets are deposited into the vault, and shares are calculated for this deposit.
2. After calculating shares, additional shares are minted for `i_guardian` and `i_vaultGuardians` using the formula `shares / i_guardianAndDaoCut`, effectively increasing the share supply.
3. This minting occurs without an equivalent increase in the vault's assets, leading to a situation where the asset-to-share ratio decreases, thus diluting the value of existing shares.
4. The dilution can be quantified by comparing the asset-to-share ratio before and after such deposits, showcasing a decrease in value per share over time as more deposits occur and additional shares are minted for guardians without asset backing.

Code

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

```
function test_share_dilution() public hasGuardian {
        // Setup: Mint and approve WETH for deposit by a user.
        uint256 depositAmount = 10 ether;
        weth.mint(depositAmount, user);
        vm.startPrank(user);
        weth.approve(address(wethVaultShares), depositAmount);

        // Capture pre-deposit state
        uint256 totalSharesBeforeDeposit = wethVaultShares.totalSupply
            ();
        uint256 guardianSharesBefore = wethVaultShares.balanceOf(
            guardian);
        uint256 vaultGuardianSharesBefore = wethVaultShares.balanceOf(
            address(vaultGuardians));

        // Perform deposit
        uint256 userShares = wethVaultShares.deposit(depositAmount,
            user);

        // Capture post-deposit state
        uint256 totalSharesAfterDeposit = wethVaultShares.totalSupply()
            ;
        uint256 guardianSharesAfter = wethVaultShares.balanceOf(
            guardian);
        uint256 vaultGuardianSharesAfter = wethVaultShares.balanceOf(
            address(vaultGuardians));

        // Calculate the shares minted for guardian and vault guardians
        uint256 sharesMintedForGuardian = guardianSharesAfter -
            guardianSharesBefore;
        uint256 sharesMintedForVaultGuardians =
            vaultGuardianSharesAfter - vaultGuardianSharesBefore;

```

```
25          // Asserts to demonstrate the finding
26          assertGt(totalSharesAfterDeposit, totalSharesBeforeDeposit +
               userShares);
27          assertTrue(sharesMintedForGuardian > 0);
28          assertTrue(sharesMintedForVaultGuardians > 0);
29          assertEq(
30              totalSharesAfterDeposit,
31              totalSharesBeforeDeposit + userShares +
                   sharesMintedForGuardian + sharesMintedForVaultGuardians
32          );
33
34          vm.stopPrank();
35      }
```

Recommended Mitigation: To address this issue, it's recommended to revise the fee structure to ensure that all shares are backed by equivalent assets. Two potential approaches are:

1. Fee Deduction from Deposited Assets: Calculate the guardian and vault guardians' fees based on the deposited assets before converting these assets into shares. This approach ensures that all minted shares are backed by assets.

```
1   function deposit(uint256 assets, address receiver)
2           public
3           override(ERC4626, IERC4626)
4           isActive
5           nonReentrant
6           returns (uint256)
7       {
8   +       uint256 guardianCut = assets / i_guardianAndDaoCut;
9   +       uint256 daoCut = assets / i_guardianAndDaoCut;
10  +       uint256 userDeposit = assets - (guardianCut + daoCut);
11  +       IERC20(asset()).transfer(i_guardian, guardianCut);
12  +       IERC20(asset()).transfer(i_vaultGuardians, daoCut);
13
14  +       if (userDeposit > maxDeposit(receiver)) {
15  +           revert VaultShares__DepositMoreThanMax(userDeposit,
        maxDeposit(receiver));
16  -       if (assets > maxDeposit(receiver)) {
17  -           revert VaultShares__DepositMoreThanMax(assets, maxDeposit(
        receiver));
18          }
19
20  +       uint256 shares = previewDeposit(userDeposit);
21  +       _deposit(_msgSender(), receiver, userDeposit, shares);
22  -       uint256 shares = previewDeposit(assets);
23  -       _deposit(_msgSender(), receiver, assets, shares);
24
25  -       _mint(i_guardian, shares / i_guardianAndDaoCut);
26  -       _mint(i_vaultGuardians, shares / i_guardianAndDaoCut);
27
```

```
+        _investFunds(userDeposit);
-        _investFunds(assets);
         return shares;
     }
```

2. Separate Fee Allocation: Allocate a separate pool of assets specifically for guardian and vault guardian fees. This pool could be funded by a transparent fee structure applied to transactions within the vault. Shares minted for guardians and vault guardians would then be backed by assets from this pool, preserving the integrity of the asset-to-share ratio for the main vault.

Low

[L-1] `VaultGuardians__UpdatedStakePrice` event emits the `newStakePrice` in place of the `oldStakePrice`

Description: In the `VaultGuardians` contract, the event, `VaultGuardians__UpdatedStakePrice` includes two uint256 parameters when emitted, `oldStakePrice` and `newStakePrice`. However, when the `VaultGuardians::updateGuardianStakePrice()` function is called, updating the stake price, the two values emitted are both the `newStakePrice`.

Impact: This misrepresentation of data can to confusion amongst users and a perceived lack of transparency about the Vault Guardians system. Third-party services that rely on events data for analysis or triggering other contracts may make erroneous assessments or decisions based on incorrect data.

Proof of Concept:

Code

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

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

    function test_VaultGuardians__UpdatedStakePrice_event_emits_new_price_twice() public {
        uint256 oldStakePrice = 9;
        uint256 newStakePrice = 10;
        vm.startPrank(vaultGuardians.owner());
        vaultGuardians.updateGuardianStakePrice(oldStakePrice);
        vm.expectEmit(address(vaultGuardians));
        emit VaultGuardians__UpdatedStakePrice(newStakePrice, newStakePrice);
        vaultGuardians.updateGuardianStakePrice(newStakePrice);
        vm.stopPrank();
```

```
12      }
```

Recommended Mitigation: Follow the Checks, Effects and Interactions (CEI) pattern.

```
1       function updateGuardianStakePrice(uint256 newStakePrice) external
            onlyOwner {
2   -       s_guardianStakePrice = newStakePrice;
3           emit VaultGuardians__UpdatedStakePrice(s_guardianStakePrice,
                newStakePrice);
4   +       s_guardianStakePrice = newStakePrice;
5       }
```

[L-2] `VaultGuardians::updateGuardianAndDaoCut()` function incorrectly emits `VaultGuardians__UpdatedStakePrice` when it should be emitting `VaultGuardians__UpdatedFee`

Description: When the `VaultGuardians::updateGuardianAndDaoCut()` function is called, the `VaultGuardians__UpdatedStakePrice` event is emitted, which should not be the case because the stake price is not being updated in this function. The function that should be emitted is `VaultGuardians__UpdatedFee`

Impact: This misrepresentation of data can to confusion amongst users and a perceived lack of transparency about the Vault Guardians system. Third-party services that rely on events data for analysis or triggering other contracts may make erroneous assessments or decisions based on incorrect data.

Proof of Concept:

Code

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

```
1   event VaultGuardians__UpdatedStakePrice(uint256 oldStakePrice, uint256
        newStakePrice);
2   .
3   .
4   .
5       function test_updateGuardianAndDaoCut_emits_wrong_event() public {
6           uint256 newCut = 10;
7           vm.startPrank(vaultGuardians.owner());
8           vm.expectEmit(address(vaultGuardians));
9           emit VaultGuardians__UpdatedStakePrice(newCut, newCut);
10          vaultGuardians.updateGuardianAndDaoCut(newCut);
11          vm.stopPrank();
12      }
```

Recommended Mitigation: Emit the `VaultGuardians__UpdatedFee` event instead of the `VaultGuardians__UpdatedStakePrice` event. It is also recommended to follow the Checks, Effects and Interactions (CEI) pattern.

```
    function updateGuardianAndDaoCut(uint256 newCut) external onlyOwner
        {
+       emit VaultGuardians__UpdatedFee(s_guardianAndDaoCut, newCut);
        s_guardianAndDaoCut = newCut;
-       emit VaultGuardians__UpdatedStakePrice(s_guardianAndDaoCut,
    newCut);
    }
```

[L-3] LINK Token Vault name and symbol are incorrectly assigned names meant for USDC vault in `VaultGuardiansBase::becomeTokenGuardian()` function

Description: The Vault Guardians system allows pre-approved tokens that are set in the constructor to be used as the underlying assets in the `VaultShares` contract. These tokens are `WETH`, `USDC`, and `LINK`, and each have a corresponding abstract contract specifying their static data, ie Vault name and symbol.

When a guardian calls the `VaultGuardiansBase::becomeTokenGuardian` function, they must pass in the address of a pre-approved token. If they pass the address for `i_tokenTwo`, which as stated in the natspec/documentation for the `AStaticTokenData` contract, is "Intended to be LINK", the symbol and name assigned to the created `VaultShares` contract are not the ones associated with LINK from the `AStaticTokenData` contract, but are in fact the ones associated with USDC from the `AStaticUSDCData` contract.

Impact: This misrepresentation of data can to confusion amongst users and a perceived lack of transparency about the Vault Guardians system. Third-party services that rely on the vault name or symbol for analysis or triggering other contracts may make erroneous assessments or decisions based on incorrect data.

Proof of Concept:

The following is from `AStaticTokenData`:

```
    // Intended to be LINK
    IERC20 internal immutable i_tokenTwo;
    string public constant TOKEN_TWO_VAULT_NAME = "Vault Guardian LINK"
        ;
    string public constant TOKEN_TWO_VAULT_SYMBOL = "vgLINK";
```

Code

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

```
    function test_link_vault_has_incorrect_name_and_symbol() public
        hasGuardian {
        address tokenTwoAddress = address(vaultGuardians.getTokenTwo())
            ;
        console.log("tokenTwoAddress:", tokenTwoAddress);
        console.log("linkAddress:", linkAddress);
        assertEq(tokenTwoAddress, linkAddress);

        // mint tokens to guardian
        link.mint(mintAmount, guardian);

        // create token vault
        vm.startPrank(guardian);
        link.approve(address(vaultGuardians), mintAmount);
        address linkVaultAddress = vaultGuardians.becomeTokenGuardian(
            allocationData, link);
        VaultShares linkVaultShares = VaultShares(linkVaultAddress);
        vm.stopPrank();

        string memory actualLinkVaultName = linkVaultShares.name();
        string memory actualLinkVaultSymbol = linkVaultShares.symbol();
        string memory expectedLinkVaultName = vaultGuardians.
            TOKEN_TWO_VAULT_NAME();
        string memory expectedLinkVaultSymbol = vaultGuardians.
            TOKEN_TWO_VAULT_SYMBOL();
        console.log("actualLinkVaultName:", actualLinkVaultName);
        console.log("actualLinkVaultSymbol:", actualLinkVaultSymbol);
        console.log("expectedLinkVaultName:", expectedLinkVaultName);
        console.log("expectedLinkVaultSymbol:", expectedLinkVaultSymbol
            );
        assert(keccak256(abi.encodePacked(actualLinkVaultName)) !=
            keccak256(abi.encodePacked(expectedLinkVaultName)));
        assert(
            keccak256(abi.encodePacked(actualLinkVaultSymbol)) !=
                keccak256(abi.encodePacked(expectedLinkVaultSymbol))
        );
    }
```

Recommended Mitigation: Modify the `VaultGuardiansBase::becomeTokenGuardian` function so that the `i_tokenTwo` vault name and symbol use `TOKEN_TWO_VAULT_NAME` and `TOKEN_TWO_VAULT_SYMBOL` from `AStaticTokenData`.

```
 } else if (address(token) == address(i_tokenTwo)) {
            tokenVault = new VaultShares(
                IVaultShares.ConstructorData({
                    asset: token,
-                   vaultName: TOKEN_ONE_VAULT_NAME,
-                   vaultSymbol: TOKEN_ONE_VAULT_SYMBOL,
+                   vaultName: TOKEN_TWO_VAULT_NAME,
+                   vaultSymbol: TOKEN_TWO_VAULT_SYMBOL,
```

```
                    guardian: msg.sender,
                    allocationData: allocationData,
                    aavePool: i_aavePool,
                    uniswapRouter: i_uniswapV2Router,
                    guardianAndDaoCut: s_guardianAndDaoCut,
                    vaultGuardians: address(this),
                    weth: address(i_weth),
                    usdc: address(i_tokenOne)
                })
            );
```

[L-4] Guardian Overwrites Existing Vault in `VaultGuardiansBase::s_guardians` Mapping

Description: In the `VaultGuardiansBase` contract, when a guardian creates a new vault using the same token as an existing vault, the `s_guardians` mapping is updated to associate the guardian with the new vault, effectively overwriting the reference to the previous vault. This behavior results in the old vault becoming inaccessible via the mapping, although it still exists on the blockchain. This issue arises due to the lack of checks or restrictions against creating multiple vaults with the same token by the same guardian, leading to an unintended state where the old vault is left in a sort of limbo.

Impact: This oversight can have several consequences:

- User Confusion and Mismanagement: Users might lose track of their assets or mistakenly believe they have been lost, leading to confusion and potential mismanagement of funds.
- System Integrity: The integrity and reliability of the contract are compromised as it does not behave as expected, potentially eroding user trust in the system.

Proof of Concept:

Code

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

```
    function test_guardian_creates_new_vault() public hasGuardian
        userIsInvested {
        IVaultShares firstVault = vaultGuardians.
            getVaultFromGuardianAndToken(guardian, weth);

        // new vault
        weth.mint(mintAmount, guardian);
        vm.startPrank(guardian);
        weth.approve(address(vaultGuardians), mintAmount);
        address wethVault2 = vaultGuardians.becomeGuardian(
            allocationData);
        VaultShares wethVaultShares2 = VaultShares(wethVault2);
```

```
10          vm.stopPrank();
11
12          IVaultShares secondVault = vaultGuardians.
                getVaultFromGuardianAndToken(guardian, weth);
13
14          assert(firstVault != secondVault);
15
16          // guardian quits second vault
17          vm.startPrank(guardian);
18          wethVaultShares2.approve(address(vaultGuardians), type(uint256)
                .max);
19          vaultGuardians.quitGuardian();
20          vm.stopPrank();
21
22          IVaultShares noVault = vaultGuardians.
                getVaultFromGuardianAndToken(guardian, weth);
23          console.log("noVault:", address(noVault));
24
25          assertEq(address(noVault), address(0));
26
27          // users can still interact with first vault
28          vm.prank(user);
29          wethVaultShares.withdraw(1, user, user);
30      }
```

Recommended Mitigation: Consider limiting vaults of a particular token to only one per guardian.

```
1 +         if(address(getVaultFromGuardianAndToken(msg.sender, _token)) !=
      address(0)) revert();
```

[L-5] Unused named return parameter in `AaveAdapter::_aaveDivest` always returns 0

Description: `AaveAdapter::_aaveDivest` function returns `amountOfAssetReturned`, which is never used in the function.

Impact: The function will always return 0, despite divesting assets back to the vault.

Recommended Mitigation:

```
1       function _aaveDivest(IERC20 token, uint256 amount) internal returns
            (uint256 amountOfAssetReturned) {
2 -         i_aavePool.withdraw({asset: address(token), amount: amount, to:
      address(this)});
3 +         amountOfAssetReturned = i_aavePool.withdraw({asset: address(
      token), amount: amount, to: address(this)});
4       }
```

[L-6] `UniswapAdapter::UniswapInvested` and `UniswapAdapter::UniswapDivested` events should return `counterpartyAmount` as the second parameter, instead of `wethAmount`

Description: When funds facilitated by the `UniswapAdapter` are invested or divested, an event is emitted including the `tokenAmount` and `wethAmount` parameters. `wethAmount` should be `counterpartyAmount` because the WETH token may be used as the primary token and its amount would correlate to `tokenAmount`.

Impact: Incorrect labelling can cause confusion amongst users.

Proof of Concept: When the event is emitted at the end of the `UniswapAdapter::UniswapInvested` function, it includes the correct labelling.

```
1       emit UniswapInvested(tokenAmount, counterPartyTokenAmount,
            liquidity);
```

Recommended Mitigation:

```
1 -     event UniswapInvested(uint256 tokenAmount, uint256 wethAmount,
      uint256 liquidity);
2 -     event UniswapDivested(uint256 tokenAmount, uint256 wethAmount);
3 +     event UniswapInvested(uint256 tokenAmount, uint256
      counterpartyAmount, uint256 liquidity);
4 +     event UniswapDivested(uint256 tokenAmount, uint256
      counterpartyAmount);
```

Informational

[I-1] Natspec in `AStaticWethData` refers to "four tokens" when there are only 3

```
1 // The following four tokens are the approved tokens the protocol
      accepts
```

The 3 tokens used in the protocol are WETH, USDC and LINK.

[I-2] Natspec in `VaultGuardians::updateGuardianAndDaoCut` describes calculation in reverse

```
1 // @dev this value will be divided by the number of shares whenever a
      user deposits into a vault
```

The shares is actually divided by the cut. See [H-1] for issue related to this.

[I-3] `VaultGuardians::sweepErc20s` lacks access control

```
function sweepErc20s(IERC20 asset) external {
        uint256 amount = asset.balanceOf(address(this));
        emit VaultGuardians__SweptTokens(address(asset));
        asset.safeTransfer(owner(), amount);
    }
```

Consider adding an `onlyOwner` or similar access control as anyone can call this function, moving contract funds to the DAO. This is optional as the contract funds are not actually used for anything.

[I-4] `VaultGuardiansBase::s_isApprovedToken` mapping is not used anywhere

```
-   mapping(address token => bool approved) private s_isApprovedToken;
```

[I-5] `VaultGuardiansBase::InvestedInGuardian` and `VaultGuardiansBase::DinvestedFromGuardian` events are not being used anywhere

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

[I-6] `VaultGuardiansBase::_becomeTokenGuardian` includes a lot of storage reads that can be reduced

```
function _becomeTokenGuardian(IERC20 token, VaultShares tokenVault)
    private returns (address) {
        s_guardians[msg.sender][token] = IVaultShares(address(
            tokenVault));
        emit GuardianAdded(msg.sender, token);

+       uint256 stakePrice = s_guardianStakePrice;

-       i_vgToken.mint(msg.sender, s_guardianStakePrice);
+       i_vgToken.mint(msg.sender, stakePrice);
-       token.safeTransferFrom(msg.sender, address(this),
    s_guardianStakePrice);
+       token.safeTransferFrom(msg.sender, address(this), stakePrice);
-       bool succ = token.approve(address(tokenVault),
    s_guardianStakePrice);
+       bool succ = token.approve(address(tokenVault), stakePrice);
        if (!succ) {
```

```
14              revert VaultGuardiansBase__TransferFailed();
15          }
16  -       uint256 shares = tokenVault.deposit(s_guardianStakePrice, msg.
        sender);
17  +       uint256 shares = tokenVault.deposit(s_guardianStakePrice, msg.
        sender);
18          if (shares == 0) {
19              revert VaultGuardiansBase__TransferFailed();
20          }
21          return address(tokenVault);
22      }
```

[I-7] `VaultShares::onlyGuardian` modifier and `VaultShares::VaultShares__NotGuardian` error are not being used anywhere

```
1   -   error VaultShares__NotGuardian();
2   .
3   .
4   .
5   -   modifier onlyGuardian() {
6   -       if (msg.sender != i_guardian) {
7   -          revert VaultShares__NotGuardian();
8   -       }
9   -       _;
10  -   }
```

[I-8] `VaultShares::deposit`, `VaultShares::withdraw` and `VaultShares::redeem` functions could be marked external

[I-9] `VaultGuardiansBase::GUARDIAN_FEE` constant isn't used

[I-10] Lack of zero address checks

[I-11] Lack of natspec for some functions

[I-12] `VaultGuardiansBase::becomeGuardian` natspec says user has to send ETH, but function is not payable

```
1       /*
2        * @notice allows a user to become a guardian
3        * @notice they have to send an ETH amount equal to the fee, and a
             WETH amount equal to the stake price
4        * @param wethAllocationData the allocation data for the WETH vault
5        */
```

```
6       function becomeGuardian(AllocationData memory wethAllocationData)
            external returns (address) {
```

[I-13] `IInvestableUniverseAdapter` interface isn't used

[I-14] `IVaultGuardians` interface isn't used