Vault Guardians Audit Report

Prepared by Ahmad Faraz

Auditing Protocol: `Vault Guardians`

Date: July 26, 2025

Contents

1 Disclaimer

I Ahmad Faraz makes every effort to understand the protocol and identify vulnerabilities within the given timeframe but holds no responsibility for missed issues. This audit is not an endorsement of the protocol's business logic or product and focuses solely on Solidity-level vulnerabilities.

2 Risk Classification

2.1 Impact Table

Likelihood	High	Medium	Low
High	High	High/Medium	Medium
Medium	High/Medium	Medium	Medium/Low
Low	Medium	Medium/Low	Low

3 Audit Details

3.1 Scope

The audit covered the following files:

- `./src/`
- `abstract/AStaticTokenData.sol`
- `dao/VaultGuardianGovernor.sol`
- `dao/VaultGuardianToken.sol`
- `interfaces/IVaultData.sol`
- `interfaces/IVaultGuardians.sol`
- `interfaces/IVaultShares.sol`
- `interfaces/InvestableUniverseAdapter.sol`
- `protocol/VaultGuardians.sol`
- `protocol/VaultGuardiansBase.sol`
- `protocol/VaultShares.sol`
- `protocol/investableUniverseAdapters/AaveAdapter.sol`
- `protocol/investableUniverseAdapters/UniswapAdapter.sol`
- `vendor/DataTypes.sol`
- `vendor/IPool.sol`

- vendor/IUniswapV2Factory.sol
- vendor/IUniswapV2Router01.sol

4 Protocol Summary

The Vault Guardians protocol allows users to deposit ERC20 tokens into ERC4626 vaults managed by vault guardians who maximize returns through strategic allocations.

4.1 Roles

- **Vault Guardian DAO:** Controls protocol parameters and receives profit share.
- **DAO Participants:** Vote on decisions and earn rewards through VaultGuardianToken.
- **Vault Guardians:** Allocate user assets to external yield strategies.
- **Investors:** Deposit tokens to earn passive yield.

5 Executive Summary

The Vault Guardians smart contract protocol audited by Ahmad Faraz. The audit revealed a total of 5 issues, classified as follows.

Severity Level	Issue Count
High	1
Medium	1
Low	3
Total	**5**

6 Findings

6.1 High Severity

6.1.1 [H-1] Arbitrary ERC20 Sweep by External Users

- **Description:** VaultGuardians::sweepErc20s function is intended to recover stray ERC20 tokens (e.g., dust amounts) accidentally left in the contract. However, this function lacks access control, meaning any user can call it at any time. This means that malicious users can sweep funds to the owner without consent, possibly disrupting protocol operations and causing unexpected fund movements.

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

- **Impact:** Users' funds may be swept without authorization.
- **Cause:** Lack of access control on `sweep` function.
- **Recommended Mitigation:** Add `onlyOwner` modifier.

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

- **Proof of Concept:**
  1. A user performs a swap or deposit operation, leaving a token residue in the vault contract.
  2. Any external caller (malicious or accidental) calls sweepErc20s.
  3. Tokens are sent to the owner, potentially affecting protocol operations or accounting logic.

  Code:

```
function testSweepErc20s() public {
        ERC20Mock mock = new ERC20Mock();
        ERC20Mock mock = new ERC20Mock();
        mock.mint(mintAmount, msg.sender);
        console2.log(mock.balanceOf(address(msg.sender)));
        vm.prank(msg.sender);
        mock.transfer(address(vaultGuardians), mintAmount);
        console2.log(mock.balanceOf(address(vaultGuardians)));
        console2.log(mock.balanceOf(address(msg.sender)));
        vm.prank(attacker);
        vaultGuardians.sweepErc20s(mock);
        console2.log(mock.balanceOf(address(vaultGuardians)));
        console2.log("Sytem owner balance ", mock.balanceOf(
            vaultGuardians.owner())
        );
    }
```

6.2 Medium Severity

6.2.1 [M-1] Voting Timers Use Seconds Instead of Blocks

- **Description:** In `VaultGuardianGovernor::votingDelay` and `VaultGuardianGovernor::votingPeriod`, the intended return is `1 days` and `7 days`, which normally seems okay, but actually, it returns the number of seconds, `1 days == 86400` and `7 days == 604800`. On-chain data works differently, meaning that if we take an average block time of 12 seconds, this is a wrong calculation. The voting will end later than expected.

```
function votingDelay() public pure override returns (uint256) {
@>          return 1 days;
}

function votingPeriod() public pure override returns (uint256) {
 @>         return 7 days;
```

```
7 }
```

- **Impact:** Delays are inconsistent across networks.
- **Recommended Mitigation:** Use block-based timing to align with governance tools.

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

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

6.3 Low Severity

6.3.1 [L-1] Incorrect Vault Name and Symbol

- **Description:** While creating new vaults in `VaultGuardiansBase::becomeTokenGuardian`, the name and symbol of the vault for `i_tokenTwo` are mistakenly set to values intended for `i_tokenOne`. This can lead to metadata mismatch, causing confusion in off-chain systems or UI.
- **Impact:** Incorrect vault metadata for `i_tokenTwo`.
- **Recommended Mitigation:** Correct constructor input for clarity.

```
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)
    }));
}
```

6.3.2 [L-2] Ignored Return Value in Aave Withdraw

- **Description:** The `AaveAdapter::_aaveDivest` function is intended to return the amount of assets returned by Aave `IPool` interface 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.
- **Impact:** Future changes in Aave API may cause logic failure.
- **Recommended Mitigation:** Store and handle returned value.

```
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)
    });
+ return amountOfAssetReturned;
}
```

6.3.3 [L-3] Incorrect Event Emitted on Fee Update

- **Description:** The function `updateGuardianAndDaoCut` emits an event related to stake pricing instead of a fee update. This misleads off-chain indexers and analytics tools.
- **Impact:** Off-chain monitoring may log inaccurate events.
- **Recommended Mitigation:** Emit correct event for clarity.

```
function updateGuardianAndDaoCut(uint256 newCut) external onlyOwner {
    uint256 oldGuardianAndDaoCut = s_guardianAndDaoCut;
    s_guardianAndDaoCut = newCut;
    emit VaultGuardians_UpdatedFee(oldGuardianAndDaoCut, newCut);
}
```