VAULT GUARDIANS

Protocol Audit Report

Version 1.0

Cyfrin.io

July 25, 2024

Vault Guardians Audit Report

GeorgiT

Prepared by: GeorgiT

Table of Contents

Disclaimer

The GeorgiT 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
```

```
21 #-- vendor
22     #-- DataTypes.sol
23     #-- IPool.sol
24     #-- IUniswapV2Factory.sol
25     #-- 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	4
Info/Gas	5
Total	16

I've reviewed the markdown and made some corrections for grammar, typos, and formatting improvements. Here is the revised version:

High

[H-1] Lack of UniswapV2 Slippage Protection for Token Ratio in `i_uniswapRouter.addLiquidity` & `i_uniswapRouter.removeLiquidity` Allows Price Manipulation and Sandwich Attacks

Description: Setting the `amountAMin` and `amountBMin` parameters in `i_uniswapRouter.addLiquidity` & `i_uniswapRouter.removeLiquidity` to `0` in `UniswapAdapter::_uniswapInvest` and `UniswapAdapter::_uniswapDivest` allows the contract to accept any ratio of tokens when adding liquidity, regardless of how unfavorable it is. Additionally, setting the `deadline` parameter to `block.timestamp` makes the protocol vulnerable to sandwich and price manipulation attacks.

Impact: This vulnerability can result in: - An attacker (e.g., a frontrunning bot) seeing this transaction in the mempool, pulling a flashloan, and swapping on Uniswap to manipulate the price before the swap, leading to an unfavorable rate for the protocol. - The transaction being held by the node that receives it until it becomes profitable to execute.

Proof of Concept: 1. A user calls `VaultShares::deposit` for a vault with a Uniswap allocation. * This triggers `_uniswapInvest` and calls the router's `swapExactTokensForTokens` function. 2. A malicious user holds this transaction in the mempool. 3. The attacker takes out a flashloan. 4. The attacker performs a large swap on Uniswap, temporarily altering the target token's price. 5. The attacker executes the held transaction. 6. The attacker then performs a back-running transaction, swapping back the tokens to profit from the price difference.

Recommended Mitigation: Set reasonable minimum amounts for both tokens based on the current pool ratio and an acceptable slippage tolerance.

[H-2] `VaultGuardians::updateGuardianAndDaoCut` Allows Guardians & DAOs to Alter Fees Favorably

Description: The `updateGuardianAndDaoCut` function allows privileged roles (guardians and DAOs) to modify critical parameters affecting asset fee structures within the vault. Without limitations or time-locks, this function can be exploited to suddenly and drastically alter the economic balance, benefiting those with voting power.

Impact: This can lead to: * Unfair price manipulation * Financial losses for users * Erosion of trust in the protocol * Potential abuse by malicious or compromised guardians/DAOs

Recommended Mitigation: Limit the voting power of guardians and DAOs.

[H-3] `VaultGuardians::sweepErc20s` Can Be Exploited to Steal All Tokens

Description: The `sweepErc20s` function can potentially allow the owner or DAO to steal all ERC20 tokens, as it sweeps all tokens instead of just excess tokens. Furthermore, the function is not restricted by `onlyOwner` and can be called by anyone.

Impact: All tokens could be stolen from the contract.

Recommended Mitigation: Restrict the function to sweep only excess ERC20 tokens and add proper ownership checks.

[H-4] `VaultShares` Contract Lacks a Minimum Amount Check, Vulnerable to DoS Attacks

Description: The `VaultShares` contract can process only a limited amount in each call. An attacker can submit numerous withdrawals with an amount of 0, causing a Denial of Service (DoS) attack. If the governance cannot afford the gas fees to process these transactions, legitimate withdrawals may be blocked.

Impact: * Congestion of the contract with numerous small transactions * Increased gas costs for legitimate users * Potential blockage of important contract functions * Degraded performance of the entire system

Proof of Concept: An attacker, Alice, spends a significant amount on gas to submit many withdrawals of amount = 0. If governance lacks the funds to process these, the withdrawals can't be processed, effectively locking funds.

Recommended Mitigation: Set a minimum withdrawal amount.

[H-5] EIP-4626 Tokenized Vaults (`VaultShares`) Susceptible to Inflation Attacks

Description: About this issue.

Impact: * Congestion of the contract with numerous small transactions * Increased gas costs for legitimate users * Potential blockage of important contract functions * Degraded performance of the entire system

Proof of Concept: An attacker with enough funds can execute a sandwich attack on depositors by: 1. Depositing to the vault to mint shares. 2. Transferring ERC20 tokens directly to the vault. 3. Monitoring and timing deposits from other users.

Recommended Mitigation: * Use a higher precision for share representation than for asset representation. * Include virtual shares and virtual assets in the exchange rate computation.

Medium

[M-1] Race Condition in Approvals Can Cause Unexpected Loss of Funds in `UniswapAdapter::_uniswapInvest`

Description: The function approves a certain amount of tokens for the contract to spend, with a time gap between the approval and the actual swap execution. During this gap, a malicious actor could potentially execute a transaction using the approved amount, leading to unexpected outcomes.

Impact: * Unauthorized token transfers * Financial losses for users * Potential front-running attacks * Erosion of trust in the protocol

Recommended Mitigation: Use OpenZeppelin's `safeIncreaseAllowance` and `safeDecreaseAllowance` functions.

[M-2] `VaultShares::divestThenInvest` Modifier Causes High Gas Costs and Possible DoS

Description: The `VaultShares::divestThenInvest` modifier initiates two transactions, causing increased gas costs when added to a function, which could potentially lead to a DoS attack.

Impact: * Long waiting times * High gas costs * Possible DoS attacks

Recommended Mitigation: Consider alternative implementations to optimize gas usage.

Low

[L-1] Centralization Risk for Trusted Owners

Description: Contracts with owners having privileged rights can pose a risk if these owners perform malicious updates or drain funds.

[L-2] Missing Checks for `address(0)` When Assigning Values to Address State Variables

Description: It's recommended to check for `address(0)` when assigning values to address state variables.

1 Found Instance

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

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

[L-3] `VaultShares::deposit` Should Check if `i_guardianAndDaoCut` Is Not 0

Description: The `i_guardianAndDaoCut` variable is modifiable by the `VaultGuardianToken` holders. If it is set to 0, even temporarily, it can cause division by zero errors, breaking the protocol.

[L-4] The `nonReentrant` Modifier Should Precede All Other Modifiers

Description: To protect against reentrancy in other modifiers, it is best practice for the `nonReentrant` modifier to be placed before any other modifiers in a function definition.

4 Found Instances

- Found in src/protocol/VaultShares.sol Line: 144
- Found in src/protocol/VaultShares.sol Line: 181
- Found in src/protocol/VaultShares.sol Line: 193
- Found in src/protocol/VaultShares.sol Line: 210

Informational/Gas

[I-1] `Public` Functions Not Used Internally Could Be Marked `External`

Description: If a function is not used internally, it should be marked as `external` rather than **`public`**.

9 Found Instances

- Found in src/dao/VaultGuardianGovernor.sol Line: 17
- Found in src/dao/VaultGuardianGovernor.sol Line: 21
- Found in src/dao/VaultGuardianGovernor.sol Line: 27
- Found in src/dao/VaultGuardianToken.sol Line: 17
- Found in src/protocol/VaultShares.sol Line: 115
- Found in src/protocol/VaultShares.sol Line: 140
- Found in src/protocol/VaultShares.sol Line: 181
- Found in src/protocol/VaultShares.sol Line: 189
- Found in src/protocol/VaultShares.sol Line: 206

[I-2] Unused Custom Error

Description: Unused custom error definitions should be removed.

4 Found Instances

```
Found in src/protocol/VaultGuardians.sol Line: 43
Found in src/protocol/VaultGuardiansBase.sol Line: 46
Found in src/protocol/VaultGuardiansBase.sol Line: 48
Found in src/protocol/VaultGuardiansBase.sol Line: 51
```

[I-3] VaultShares::onlyGuardian Modifier Is Unused

[I-4] VaultGuardianBase::_guardianHasNonWethVaults Function Can Be Gas Optimized

[I-5] Unused Interfaces

Description: The IInvestableUniverseAdapter & IVaultGuardians interfaces are empty and unused and should likely be removed.