Lagoon Security Review

Version 1.0

Conducted by:
Bretzel, Independent Security Researcher

Table of Contents

1 About Bretzel

I am a smart contract security researcher with prior experience involving EIP-4626 with Amphor under the alias *gp-20* . Additionally, I participated to Block VI- YAcademy as *willboy* on this review on Yearn Finance. To explore a comprehensive overview of my expertise and projects, please visit my website. Feel free to connect with me via Telegram or Twitter.

2 Disclaimer

Reviews are a time, resource and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to find as many vulnerabilities as possible. Reviews can show the presence of vulnerabilities **but not their absence**.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that's not so critical.

3.2 Likelihood

- **High** - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- **Medium** - only conditionally incentivized attack vector, but still relatively likely.
- **Low** - too many or too unlikely assumptions; provides little or no incentive.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Lagoon
Repository	`https://github.com/hopperlabsxyz/vault`
Commit hash	Started at 18ee01348021abc6bd4376b68f51be98dadeb847
Resolution	See during Findings or here
Methods	Manual review

Scope

protocol/Events.sol
protocol/FeeRegistry.sol
vault/ERC7540.sol
vault/Vault.sol
vault/Silo.sol
vault/Roles.sol
vault/FeeManager.sol
vault/Events.sol
vault/Errors.sol
vault/Enums.sol
vault/Whitelistable.sol
vault/interfaces/IERC7540.sol
vault/interfaces/IERC7540Deposit.sol
vault/interfaces/IERC7540Redeem.sol
vault/interfaces/IERC7575.sol
vault/interfaces/IWETH9.sol

Issues Found

Critical risk	0
High risk	0
Medium risk	2
Low risk	3
Informational	2
Gas Optimisation	3

5 System overview

Lagoon Protocol is a decentralized platform designed for asset management, allowing asset managers to create and manage **Lagoon Vaults**. These Vaults offer non-custodial and risk-managed solutions for handling digital assets efficiently.

The protocol is built using established smart contract standards and incorporates components such as **Gnosis Safe** and **Zodiac Roles Modifier**. This combination facilitates the creation of customizable vaults tailored to various asset management needs.

Lagoon Vaults enable the implementation of diverse decentralized finance (DeFi) strategies, including asset management and yield farming. The design of the protocol allows asset managers to configure their Vaults with specific DeFi protocol whitelists, power distribution mechanisms, and fee structures to meet their particular requirements.

5.1 Core Features

At smart contract level, Lagoon Protocol is supported by a structured infrastructure that includes:

- **ERC-7540 Standard**: Establishes an asynchronous vault standard that tokenizes user shares, allowing users to deposit assets into the Vault and receive corresponding share tokens.
- **Fee Management Module**: Provides Vault creators with the ability to define, adjust, or remove various fees, including performance, management, and entry/exit fees, thereby allowing for customized economic models.

5.2 Privileged actors

- Asset Managers as `safe`: Each Lagoon Vault is associated with a Safe smart contract. This Safe acts as the primary holder of the vault's assets and is responsible for settling deposits and redeems.
- NAV Manager as `navManager` : The address is tasked with updating the `newTotalAssets` value of the vault.
- Owner as `owner` (vault part): Although not explicitly part of the `RolesStorage` struct, the Owner holds administrative control over the contract. The Owner is considered the admin and possesses the authority to modify other roles and itself. The Owner manages critical aspects of the vault, including initiating the closure of the vault, pausing or unpausing the vault the vault, disabling the whitelist, and updating fee rates (both management and performance fees).
- Owner as `owner` (protocol part): Update protocol fee rates.
- Whitelist Manager as `whitelistManager` : This address is responsible for managing the whitelist.
- User : Interact with the `Vault`. A user can make request (`requestDeposit` or `requestRedeem`) or cancelled it through `cancelRequestDeposit`. When an epoch is settled the user can `deposit` / `mint` or `redeem` / `withdraw`. Even if the `Vault` is closed, users retain the capability to `redeem` / `withdraw` their assets.

6 Findings

6.1 Medium risk

6.1.1 Edge case : Possibility to DOS user to requestDeposit and requestRedeem due to a lack of verification on controller parameter

Severity: *Medium risk*

Context: ERC7540.sol

Description:

The likelihood is very low, but it can DOS some user to use the Vault.

In the current implementation, the modifier `onlyOperator(owner)` or L328 check that the `msg.sender` is the owner OR the operator (`msg.sender`) has the permission of the owner to do an operation on his behalf. requestDeposit - owner MUST equal msg.sender unless the owner has approved the msg.sender as an operator.

There is no check on the `controller` parameter, so the `msg.sender` can put any address for `controller`. Controller is not the address that will be used to deposit assets. However, it will be used to handle the request inside the Vault. The controller can set an operator in order that someone call the function on his behalf like at L386 or at L287.

In addition, there is the invariant `OnlyOneRequestAllowed`. For each controller, there is only one pending request accepted. It's the request ID that match the current `depositEpochId` or `redeemEpochId`.

If we combine the information above, there is this extreme edge case :

1. A malicious user front-run updateNewTotalAssets. He will call `requestDeposit` or `requestRedeem` with 1 wei of value and put an address that he wants to block to `controller`.
2. `updateNewTotalAssets` will be executed, depositEpochId or redeemEpochId will increase by 2.
3. The victim wants to make a request at this moment (the timing is VERY unlucky), it will revert due to `OnlyOneRequestAllowed`.

POC:

- POC_DOS_RequestRedeem.t.txt `forge test --match-path test/POC_DOS_RequestRedeem.t.sol -vvv`

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;

import {BaseTest} from "./Base.sol";
import {IERC20Errors} from "@openzeppelin/contracts/interfaces/draft-IERC6093.sol";
import {IERC20, SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20
    .sol";
import {OnlyOneRequestAllowed} from "@src/vault/ERC7540.sol";
import {Vault} from "@src/vault/Vault.sol";
import "forge-std/Test.sol";

contract TestRequestRedeem is BaseTest {
    function setUp() public {
        enableWhitelist = false;
```

```
        setUpVault(0, 0, 0);
        dealAndApprove(user1.addr);
        uint256 balance = assetBalance(user1.addr);
        requestDeposit(balance, user1.addr);
        updateAndSettle(0);
        deposit(balance, user1.addr);
    }

    function test_dos_requestRedeem() public {
        address owner_1 = user1.addr;
        address owner_2 = user2.addr;
        address owner_3 = user3.addr;
        address controller_dos_1 = user2.addr;
        address controller_dos_2 = user3.addr;

        uint256 ownerBalance = 1;

        // Front run updateNewTotalAssets
        vm.prank(owner_1);

        requestRedeem(ownerBalance, controller_dos_1, owner_1);
        assertEq(
            ownerBalance,
            vault.pendingRedeemRequest(0, controller_dos_1),
            "owner former balance should be the controller pending Redeem request"
        );

        requestRedeem(ownerBalance, controller_dos_2, owner_1);
        assertEq(
            ownerBalance,
            vault.pendingRedeemRequest(0, controller_dos_2),
            "owner former balance should be the controller pending Redeem request"
        );

        updateNewTotalAssets(0);

        vm.prank(owner_2);
        vm.expectRevert(OnlyOneRequestAllowed.selector);
        vault.requestRedeem(ownerBalance, owner_2, owner_2);

        vm.prank(owner_3);
        vm.expectRevert(OnlyOneRequestAllowed.selector);
        vault.requestRedeem(ownerBalance, owner_3, owner_3);

    }
}
```

- POC_DOS_RequestDeposit.t.txt `forge test --match-path test/POC_DOS_RequestDeposit.t.sol -vvv`

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.26;

import {BaseTest} from "./Base.sol";
import {IERC20, SafeERC20} from "@openzeppelin/contracts/token/ERC20/utils/SafeERC20
    .sol";
```

```
import {CantDepositNativeToken, ERC7540InvalidOperator, OnlyOneRequestAllowed} from
    "@src/vault/ERC7540.sol";
import {Vault} from "@src/vault/Vault.sol";
import "forge-std/Test.sol";

contract TestRequestDeposit is BaseTest {
    function setUp() public {
        setUpVault(0, 0, 0);
        dealAndApproveAndWhitelist(user1.addr);
        whitelist(user1.addr);
        whitelist(user2.addr);
        whitelist(user3.addr);
    }

    function test_dos_requestDeposit() public {
        address owner_1 = user1.addr;
        address owner_2 = user2.addr;
        address owner_3 = user3.addr;
        address controller_dos_1 = user2.addr;
        address controller_dos_2 = user3.addr;

        uint256 ownerBalance = 1;

        // Front run updateNewTotalAssets
        requestDeposit(ownerBalance, controller_dos_1, owner_1);
        assertEq(
            ownerBalance,
            vault.pendingDepositRequest(0, controller_dos_1),
            "owner balance should be the controller pending deposit request"
        );

        requestDeposit(ownerBalance, controller_dos_2, owner_1);
        assertEq(
            ownerBalance,
            vault.pendingDepositRequest(0, controller_dos_2),
            "owner balance should be the controller pending deposit request"
        );

        updateNewTotalAssets(0);

        vm.prank(owner_2);
        vm.expectRevert(OnlyOneRequestAllowed.selector);
        vault.requestDeposit(ownerBalance, owner_2, owner_2);

        vm.prank(owner_3);
        vm.expectRevert(OnlyOneRequestAllowed.selector);
        vault.requestDeposit(ownerBalance, owner_3, owner_3);

    }
}
```

Recommendation:

The simple way is to add another modifier `onlyOperator` that this time is checking that the `msg.sender` is an operator of `controller`.

```
function requestDeposit(
        uint256 assets,
        address controller,
        address owner
    ) public payable virtual onlyOperator(owner) onlyOperator(controller)
        whenNotPaused returns (uint256)

function requestRedeem(uint256 shares, address controller, address owner) public
    virtual onlyOperator(controller) returns (uint256)
```

Resolution: Acknowledged.

6.1.2 Edge case : When close() the vault, it transfers all the underlying of the safe

Severity: *Medium risk*

Context: Vault.sol#L265

Description:

When the safe is closing the vault, it will transfer all the underlying of the safe.

As we spoked, there can be some cases where 1 safe is managing multiple vaults. These vaults, can have the same underlying assets. When one of the vault is closed. He will have all the underlying, including the amounts for the second vault. Furthermore, there is no possibility to give back the amount "stolen".

Recommendation:

Honestly, I don't have a clear recommendation. There can be two paths :

- Forced to have a 1:1 relation between a Safe and a Vault
- Add a parameter that will represent the amount to transfer

Resolution: use msg.sender instead of safe() and use totalAssets instead of asset… #142

Comments:

INFORMATIONAL Checks-effects-interactions pattern is not respected

From : security-considerations The checks-effects-interactions pattern is a common practice in smart contract design to prevent reentrancy attacks. At the end of close(), the `_getVaultStorage().state` is set to `Status.Closed` after funds have been transferred.

The state variable change should be done before the asset transfer.

Final Resolution: Resolved.

6.2 Low risk

6.2.1 Edge case : if _decimalsOffset() is set then wrong computation when calculating fees

Severity: *Low risk*

Context: FeeManager.sol#L203

Description:

In the current implementation, `_decimalOffset()` is set to 0. It means that : 10 ** _decimalsOffset() = 10 ** 0 = 1

There is no impact that `totalSupply + 1` is hard coded in this case.

However, if it comes to change, the computation will be incorrect. Has the `10 ** _decimalsOffset()` can only increase. If `totalSupply + 1` is kept, there will be fewer fees than it supposed to be.

Recommendation:

```
uint256 totalShares = totalFees.mulDiv(_totalSupply + 10 ** _decimalsOffset(), (
    totalAssets() - totalFees) + 1, Math.Rounding.Ceil);
```

Like here

Resolution: Resolved - add _decimalsOffset in shares convertion in FeeManager #124

6.2.2 Missing modifier onlyOpen for for Vault.requestDeposit() and Vault.updateNewTotalAssets()

Severity: *Low risk*

Context: Vault.sol#L130

Description:

Missing the modifier for Vault.requestDeposit(): The modifier is on requestRedeem() but not on requestDeposit() L130 - requestDeposit() L144

You can still recover your request by using `cancelDeposit` only if no one called `updateNewTotalAssets` .

Missing the modifier for Vault.updateNewTotalAssets(): All the other functions have this modifier so in the same logic it can have it. Furthermore, if `updateNewTotalAssets` is called then it's not possible to call `cancelDeposit` to recover your request. It will revert due to this check L313.

Recommendation:

Vault.requestDeposit(): Add the modifier to these functions.

Vault.updateNewTotalAssets() Add the modifier to the function.

Resolution: Resolved - verify in updateNewTotalAssets that the vault is not Closed #123

6.2.3 Missing modifier whenNotPaused and reorganize it

Severity: *Low risk*

Context: Vault.sol and ERC7540.sol

Description:

- There is a path where the modifier is missing for Vault.redeem: The modifier is not called when entering this path L364.
- Put the modifier in Vault.requestDeposit instead of ERC7540.requestDeposit: For requestRedeem, the modifier is put in Vault.sol with other modifier.
- Confusion i the comments of ERC7540.deposit() and ERC7540.mint(): `deposit()` : L241 - L249

`mint()` : L274 - L282

The share is an ERC20 Pausable so no need to have the modifier whenNotPaused but the comments can be confusing

Recommendation:

- There is a path where the modifier is missing for Vault.redeem: If the modifier is added at `redeem()` then the modifier in ERC7540._redeem can be removed.
- Put the modifier in Vault.requestDeposit instead of ERC7540.requestDeposit: - Put the modifier in Vault.requestDeposit instead of ERC7540.requestDeposit
- Confusion i the comments of ERC7540.deposit() and ERC7540.mint(): Update comments

Resolution:

After deeper review this path L364 is actually protected by the modifier thanks to erc20Pausable, I am going to add a comment on this.

Resolved - add explanation of how various functions are subject to pausability a… #128

6.3 Informational

6.3.1 Missing events for roles only functions

Severity: *Informational*

Context: Accross the codebase

Description:

Several key actions are defined without event declarations. Roles only functions that change critical parameters can emit events to record these changes on-chain for off-chain monitors/tools/interfaces.

Sources:

- Spearbit review of Paladin Quest - 5.5.1

Recommendation:

Add events to all roles functions that change critical parameters.

- FeeRegistry.sol
  - updateProtocolFeeReceiver
  - setProtocolRate
  - setCustomRate
  - cancelCustomRate
- Roles.sol
  - updateWhitelistManager
  - updateNAVManager
  - updateFeeReceiver
- Whitelistable.sol
  - deactivateWhitelist
- FeeManager.sol
  - updateRates
  - _setHighWaterMark ?

Resolution: Resolved - add events for storage updates #126

6.3.2 Reinitialize custom rate of the vault in cancelCustomRate()

Severity: *Informational*

Context: FeeRegistry.sol#L66

Description:

In the FeeRegistry contract, the cancelCustomRate() function deactivates a custom rate for a specific vault by setting `_getFeeRegistryStorage().customRate[vault].isActivated` to false. However, it does not reset the corresponding `_getFeeRegistryStorage().customRate[vault].rate`. This oversight can lead to confusion when interacting with the customRate() function.

Furthermore, when you call `setCustomRate()` you need to pass a `rate` value. In any case, you need to give a new rate and can't reuse the older one.

Recommendation:

Add `_getFeeRegistryStorage().customRate[vault].rate = 0` inside `cancelCustomRate()`

Resolution:

Resolved by removing `customRate()` and merge `setCustomRate()` and `disableCustom()` rate in one function - Resolved - rmv customRate; and merge setCustomRate and disableCustom rate in one fx #133

6.4 Gas Optimization

6.4.1 Optimization on Whitelistable.sol

Severity: *Gas Optimization*

Context: Whitelistable.sol

Description: Optimize Loop

Sources:

- Spearbit review of Paladin Quest - 5.4.14
- Paladin Quest - Example Loop

Recommendation:

```
function addToWhitelist(address[] memory accounts) external onlyWhitelistManager {
        WhitelistableStorage storage $ = _getWhitelistableStorage();
    uint256 length = accounts.length;

        if ($.root != 0) revert MerkleTreeMode();

        if(length == 0) revert Errors.EmptyParameters();

        for(uint256 i; i < length;){
            $.isWhitelisted[accounts[i]] = true;
            emit WhitelistUpdated(accounts[i], true);

            unchecked{ ++i; }
        }
    }
```

Same logic for `revokeFromWhitelist`

Resolution: Resolved - Refacto/opti #130

6.4.2 Optimization on FeeRegistry.sol

Severity: *Gas Optimization*

Context: FeeRegistry.sol

Description: Reduce slots of the struct FeeRegistryStorage

`protocolRate` use an **`uint256`** it means that it take one slot. It's possible to use an uint16 like for rate or managementRate

Recommendation: Use **`uint16`**

Resolution: Resolved - Refacto/opti #130

6.4.3 Optimization on FeeManager.sol - 1

Severity: *Gas Optimization*

Context: FeeManager.sol

Description:

Reuse the local variable _totalAssets instead of totalAsset() inside_calculateFees()

A `_totalAssets` local variable is initialized at L189. It can be reused instead of using totalAssets()

Recommendation:

```
uint256 totalShares = totalFees.mulDiv(_totalSupply + 1, (_totalAssets - totalFees)
    + 1, Math.Rounding.Ceil);
```

Resolution: Resolved - Refacto/opti #130

6.4.4 Optimization on FeeManager.sol - 2

Severity: *Gas Optimization*

Context: FeeManager.sol

Description:

Changing public constant variables to non-public can save gas

Several constants are public and thus have a getter function. It is unlikely for these values to be called from the outside, therefore it is not necessary to make them public

Recommendation:

Make constants that do not need to be accessible from the outside private.

```
uint256 constant private ONE_YEAR = 365 days;
uint256 constant private BPS_DIVIDER = 10_000; // 100 %
```

Source: Spearbit review of Paladin Quest - 5.4.1

Resolution: Acknowledged.