Brrito Security Review

Pashov Audit Group

Conducted by: Said, immeas

January 10th 2024 - January 13th 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 **brr-eth** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About brrETH

brrETH is a yield-bearing ETH derivative built on Compound III's Base WETH market.

brrETH is easy to use and understand: deposit ETH, receive brrETH. Your brrETH can be redeemed at any time for the amount of ETH you originally deposited, plus any interest accrued.

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 **- 694d7c6277669ffdbc947e21ff651a138a56d883**

fixes review commit hash **- 3157ddf424572f132f2c4b60ec42965a4ae33721**

Scope

The following smart contracts were in scope of the audit:

- `BrrETH`
- `BrrETHRedeemHelper`
- `interfaces/**`

7. Executive Summary

Over the course of the security review, Said, immeas engaged with Brrito to review brrETH. In this period of time a total of **9** issues were uncovered.

Protocol Summary

Protocol Name	brrETH
Repository	https://github.com/brritoxyz/brr-eth
Date	January 10th 2024 - January 13th 2024
Protocol Type	Yield-bearing ETH derivative

Findings Count

Severity	**Amount**
Medium	3
Low	6
Total Findings	**9**

Summary of Findings

ID	Title	Severity	Status
[M-01]	The result from getSwapOutput is inaccurate to be used as the minOutput in the swap operation	Medium	Resolved
[M-02]	harvest function is susceptible to sandwich attacks and any unexpected market events	Medium	Acknowledged
[M-03]	Initial griefing attack possible	Medium	Resolved
[L-01]	Unsupported ERC4626 functions should revert the call rather than silently succeeding	Low	Resolved
[L-02]	harvest should be triggered before changing Comet Rewards	Low	Resolved
[L-03]	approveTokens should be triggered after changing Comet Rewards and Router	Low	Resolved
[L-04]	lack of slippage protection for deposit and redeem	Low	Resolved
[L-05]	Insufficient Input Validation	Low	Resolved
[L-06]	previous approvals not revoked	Low	Resolved

8. Findings

8.1. Medium Findings

[M-01] The result from `getSwapOutput` is inaccurate to be used as the `minOutput` in the swap operation

Severity

Impact: Low, because it decrease the `minOutput` by 0.02%

Likelihood: High, because it always happened when `harvest` is called.

Description

When `harvest` called, it will calculate `quote` amount that will be used for `minOutput` in the swap operation by calling `router.getSwapOutput`

```
function harvest() external {
    // ...
    // Fetching the quote onchain means that we're subject to front/back-running
    // but the
    // assumption is that we will harvest so frequently that the rewards won't
    // justify the effort.
>>  (uint256 index, uint256 quote) = router.getSwapOutput(
        keccak256(abi.encodePacked(rewardConfig.token, _WETH)),
        rewards
    );

    // `swap` returns the entire WETH amount received from the swap.
    uint256 supplyAssets = router.swap(
        rewardConfig.token,
        _WETH,
        rewards,
>>      quote,
        index,
        // Receives half of the swap fees
        //(the other half remains in the router contract for the protocol).
        feeDistributor
    );
    // ...
}
```

Here is the calculation inside `router.getSwapOutput`, it will compute the output after deducting the fee.

```
function getSwapOutput(
        bytes32 pair,
        uint256 input
    ) external view returns (uint256 index, uint256 output) {
        IPath[][] memory routes = _routes[pair];
        uint256 routesLength = routes.length;

        unchecked {
            for (uint256 i = 0; i < routesLength; ++i) {
                IPath[] memory route = routes[i];
                uint256 routeLength = route.length;
                uint256 quoteValue = input;

                for (uint256 j = 0; j < routeLength; ++j) {
                    quoteValue = route[j].quoteTokenOutput(quoteValue);
                }

                if (quoteValue > output) {
                    index = i;
                    output = quoteValue;
                }
            }
        }

>>      output = output.mulDiv(_FEE_DEDUCTED, _FEE_BASE);
    }
```

However, within the `swap` operation inside the router, the check for `minOutput` is performed before deducting the fee.

```
function _swap(
        address inputToken,
        address outputToken,
        uint256 input,
        uint256 minOutput,
        address outputRecipient,
        uint256 routeIndex,
        address referrer
    ) private returns (uint256 output) {
        IPath[] memory route = _routes[
            keccak256(abi.encodePacked(inputToken, outputToken))
        ][routeIndex];
        uint256 routeLength = route.length;
        output = outputToken.balanceOf(address(this));

        for (uint256 i = 0; i < routeLength; ) {
            input = route[i].swap(input);

            unchecked {
                ++i;
            }
        }

        // The difference between the balances before/after the swaps is the
        // canonical output.
        output = outputToken.balanceOf(address(this)) - output;

>>      if (output < minOutput) revert InsufficientOutput();

        unchecked {
            uint256 originalOutput = output;
            output = originalOutput.mulDiv(_FEE_DEDUCTED, _FEE_BASE);

            // Will not overflow since `output` is 99.98% of `originalOutput`.
            uint256 fees = originalOutput - output;

            outputToken.safeTransfer(outputRecipient, output);

            // If the referrer is non-zero, split 50% of the fees
            //(rounded down) with the referrer.
            // The remainder is kept by the contract which can later be
            // withdrawn by the owner.
            if (referrer != address(0) && fees > 1) {
                // Will not overflow since `fees` is 2 or greater.
                outputToken.safeTransfer(referrer, fees / 2);
            }

            emit Swap(inputToken, outputToken, routeIndex, output, fees);
        }
    }
```

Using `quote` result from `getSwapOutput` means it will compare the `output` using value less than it should be.

Recommendations

Adjust the `swap` functionality inside the router to check `minOutput` after deducting the fee or adjust the `qoute` result so it is reflect the output amount before fee is deducted.

[M-02] `harvest` function is susceptible to sandwich attacks and any unexpected market events

Severity

Impact: High, Because attackers can sandwich the operation and steal `swap` value, or in unexpected market events, the `swap` could result in an unexpectedly low value.

Likelihood: Medium, Because the attack vector is quite common and well-known, and price volatility is typical for non-stable coin tokens.

Description

While acknowledged by the protocol team, using `getSwapOutput` to calculate the minimum output of the swap on-chain is still not recommended under any circumstance or assumption. This method is not only vulnerable to sandwich attacks but also susceptible to any market events, such as rapid price changes.

Besides that, the assumption that harvest will always be callable is not correct, as `supply` functionality to `_COMET` can be paused, causing the calls to revert. In the unlikely, but possible, event that the compound pauses the WETH pool, interest would still accrue, and the reward amount would build up, becoming large enough for sandwich attacks to become feasible.

```
function harvest() external {
        // ...

        // Fetching the quote onchain means that we're subject to
        // front/back-running but the
        // assumption is that we will harvest so frequently that the rewards
        // won't justify the effort.
        // @audit - quote here, already deducted by fee, while the minOutput
        // check at swap, is step before fees are deducted
>>      (uint256 index, uint256 quote) = router.getSwapOutput(
            keccak256(abi.encodePacked(rewardConfig.token, _WETH)),
            rewards
        );

        // `swap` returns the entire WETH amount received from the swap.
>>      uint256 supplyAssets = router.swap(
            rewardConfig.token,
            _WETH,
            rewards,
            quote,
            index,
            // Receives half of the swap fees
            //(the other half remains in the router contract for the protocol).
            feeDistributor
        );

       // ...
    }
```

Recommendations

Consider putting the minimum output as a parameter inside the `harvest` function, and if this function is planned to be frequently called by bots, it could be restricted so that only certain roles can invoke it.

[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 solady library. However, doing this attack can cause some weird 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

```
address bob = makeAddr("bob");

    function testInitialSupplyAttack() public {
        // attacker starts with 13 ether
        _getCWETH(13e18 + 6);

        // initial small deposit
        vault.deposit(11,address(this));
        assertEq(10,vault.balanceOf(address(this)));

        // large deposit to inflate the exchange rate
        _COMET.safeTransfer(address(vault),11e18-9);

        // share price is not 1e18 assets
        assertEq(1e18,vault.convertToAssets(1));

        // boilerplate to get cWETHv3
        deal(_WETH,bob,1e18 + 3);
        vm.startPrank(bob);
        _WETH.safeApprove(_COMET,1e18+3);
        IComet(_COMET).supply(_WETH, 1e18+3);
        _COMET.safeApproveWithRetry(address(vault), type(uint256).max);

        // victim deposits into the vault
        vault.deposit(1e18+1,bob);
        // due to exchange rate gets 0 shares
        assertEq(0,vault.balanceOf(bob));
        vm.stopPrank();

        vault.redeem(10, address(this), address(this));
        console.log("exchange rate",vault.convertToAssets(1));
        console.log("_COMET.balanceOf(address(vault))",_COMET.balanceOf(address
          (vault)));
        console.log("_COMET.balanceOf(address(attacker))",_COMET.balanceOf
          (address(this)));
    }
```

with the output:

```
Logs:
  exchange rate 1090909090909090909
  _COMET.balanceOf(address(vault)) 1090909090909090908
  _COMET.balanceOf(address(attacker)) 12909090909090909091
```

As you can see the attacker needs to pay `0.1 eth` for the attack. But they have effectively locked the victims `1 eth` 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 his tokens.

Recommendations

Consider mitigating this with an initial deposit of a small amount.

8.2. Low Findings

[L-01] Unsupported ERC4626 functions should revert the call rather than silently succeeding

`BrrETH` is designed to not fully comply with ERC4626. However, the unsupported functions (`mint`, `withdraw`, `previewWithdraw`, `previewMint`, `maxWithdraw`, and `maxMint`) are overridden with empty functions rather than reverting. If a third party or integrator works with `BrrETH`, the call to the function will silently succeed. It is considered best practice to revert unsupported functions rather than override them with empty functions.

[L-02] `harvest` should be triggered before changing Comet Rewards

Changing Comet Rewards without triggering the `harvest` function could result in the reward token from the previous Comet Rewards contract getting stuck inside `BrrETH`. It is recommended to have an additional boolean parameter that provides the option for the caller to trigger the `harvest` function inside `setCometRewards`, or leave a note/comment to always call `harvest` before changing set comet rewards if possible.

[L-03] `approveTokens` should be triggered after changing Comet Rewards and Router

Changing Comet Rewards and Router without immediately triggering the `approveTokens` function could result in a revert period when `harvest` is triggered due to a lack of approval. Consider immediately calling `approveTokens` inside the `setCometRewards` and `setRouter` functions.

[L-04] lack of slippage protection for `deposit` and `redeem`

`deposit`'s calculated shares and `redeem`'s calculated assets for users highly depend on the current total assets and total supply inside the `BrrETH` vault, which can change in value, impacting the result of shares/assets calculation. Consider to create a router/helper that wraps the `deposit` function, providing slippage checks. Additionally, add the slippage check for `redeem` inside the `BrrETHRedeemHelper`.

[L-05] Insufficient Input Validation

`setRewardFee` allows the owner to set `rewardFee` greater than `_FEE_BASE`, which could lead to unexpected behavior. The call will revert because `supplyAssets -= fees` will underflow, causing the supply to `COMET` to fail due to an excessively high value in `supplyAssets`. Besides that, having a maximum cap for the reward fee is generally a good practice to minimize the centralization risk issue.

Discussion

Pashov Audit Group: `rewardFee` is now checked to ensure it cannot be greater than `_FEE_BASE`. However, we still recommend implementing a check against the maximum fee percentage to minimize centralization risk.

[L-06] previous approvals not revoked

When constructing the contract max approval is set for the router for the Comet reward token:

```
// Enable the router to swap our Comet rewards for WETH.
        rewardConfig.token.safeApproveWithRetry(
            address(router),
            type(uint256).max
        );
```

However, compound governance can change the reward token on `Comet`:

```
function setRewardConfig(address comet, address token) external {
        setRewardConfigWithMultiplier(comet, token, FACTOR_SCALE);
    }
```

And `owner` can change the `router` on `BrrETH`:

```
function setRouter(address _router) external onlyOwner {
        if (_router == address(0)) revert InvalidRouter();

        router = IRouter(_router);

        emit SetRouter(_router);
    }
```

If any of these are changed the previous approval will still be left.

Consider adding a call to revoke/set any approvals, callable only by `owner`, similar to what Comet has:

```
function approveThis
      (address manager, address asset, uint amount) override external {
        if (msg.sender != governor) revert Unauthorized();

        ERC20(asset).approve(manager, amount);
    }
```

That way `owner` can modify/revoke any approvals needed. Since `setRouter` is under the control of the protocol also consider including a call in it to reset the approval to the previous router already there.