Berachain Honey Security Review

Auditors

Noah Marconi, Lead Security Researcher

0xLadboy, Security Researcher

Rvierdiiev, Associate Security Researcher

Report prepared by: Lucas Goiriz

November 25, 2024

Contents

1 About Spearbit **2**

2 Introduction **2**

3 Risk classification **2**
3.1 Impact . . . 2
3.2 Likelihood . . . 2
3.3 Action required for severity levels . . . 2

4 Executive Summary **3**

5 Findings **4**
5.1 Medium Risk . . . 4
5.1.1 V2: `HoneyFactoryReader` required collateral calculation is incorrect . . . 4
5.1.2 V2: Basketmode provides insufficient protection in severe depegs . . . 4
5.1.3 V2: Single bad collateral block `HoneyFactory#mint` in basket mode forever . . . 5
5.1.4 V2: Liquidated and 0 balance depegged asset causes basket mode . . . 7
5.1.5 V2: `HoneyFactory.mint()` may spend more funds than was expected during the basket mode switch . . . 7
5.2 Low Risk . . . 8
5.2.1 V2: `setPriceFeed` should be managed by `DEFAULT_ADMIN_ROLE` . . . 8
5.2.2 V2: Asset settings made prior to vault creation are silently reset . . . 8
5.2.3 V2: Mint and Redeem rates initialize to 0 allowing for 100% fee . . . 9
5.2.4 V2: Restrict `priceFeedMaxDelay`, `lowerPegOffsets` and `upperPegOffsets` to reasonable ranges . . . 9
5.2.5 V2: Precision loss in `goodShares` refund logic in `HoneyFactory.liquidation` . . . 9
5.2.6 V2: Frontrunning `HoneyFactory#recapitalize` via donating can DoS `HoneyFactory#mint` if user approve exact token amount . . . 10
5.2.7 V2: `HoneyFactory#mint` contain division by zero error when basket mode is activated. . . . 12
5.2.8 V2: Caps are not checked after redemption in normal mode . . . 13
5.2.9 Consider introduce mint / redeem timelock to further mitigate cross-asset arbitrage . . . 13
5.3 Gas Optimization . . . 14
5.3.1 V2: Do not `_mint` or `_redeem` when amount is 0 . . . 14
5.3.2 V2: Liquidate should fail when there are 0 `badShares` . . . 14
5.3.3 V2: Returning weights as a numerator and denominator can eliminate one loop . . . 14
5.3.4 V2: Store `registrationIndex` to save on gas . . . 14
5.3.5 Consider immutables for gas efficiency . . . 15
5.4 Informational . . . 15
5.4.1 V2: Depegged assets received on redeem even when Honey is fully recapitalized . . . 15
5.4.2 V2: Move test functions to a test only contract . . . 15
5.4.3 V2: Address pending TODOs and TBDs . . . 16
5.4.4 V2: Consider adding more testing for liquidation . . . 16
5.4.5 V2: State update function missing event emission . . . 16
5.4.6 Consider adding storage gap in upgradeable contracts . . . 17
5.4.7 Validate all token addresses have code or make use of OZ's `safeTransfer` . . . 17
5.4.8 Consider initialization of `PausableUpgradeable` inside `CollateralVault` and `VaultAdmin` contracts . . . 17
5.4.9 Consider making `mintRates` and `redeemRates` public . . . 18
5.4.10 Consider reverting transaction early with explicit check and revert reason when a specific vault is paused . . . 18

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Berachain is an EVM-identical L1 turning liquidity into security powered by Proof Of Liquidity.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of contracts-monorepo according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	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 loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 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

4 Executive Summary

Over the course of 7 days in total, Berachain engaged with Spearbit to review the contracts-monorepo protocol. In this period of time a total of **29** issues were found.

Summary

Project Name	Berachain
Repository	contracts-monorepo
Commit	17178a...8157
Type of Project	Lending, Yield
Audit Timeline	Oct 23 to Nov 1
Fix period	Nov 1

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	5	3	2
Low Risk	9	6	3
Gas Optimizations	5	1	4
Informational	10	6	4
Total	**29**	**16**	**13**

5 Findings

5.1 Medium Risk

5.1.1 V2: `HoneyFactoryReader` required collateral calculation is incorrect

Severity: Medium Risk

Context: HoneyFactoryReader.sol#L75-L78

Description: `HoneyFactoryReader.previewRequiredCollateral()` function provides amounts that users should pay to mint `exactHoneyAmount`. It calculates amounts for each asset in the basket that user should provide.

```
uint256 shares = exactHoneyAmount * weights[j] / mintRate;
uint256 amount = vault.previewMint(shares);
// Convert to asset decimals and then to 18 decimals.
amount = Utils.changeDecimals(amount, 18, assetDecimal);
res[j] = Utils.changeDecimals(amount, assetDecimal, 18);
```

In this calculation `amount` is returned in vault's asset decimals. So for example, if we have a USDC vault, then the amount is returned in 10^6 precision.

Then later, we see that amount is converted from 18 decimals to `assetDecimal` and then back to 18 decimals. This looks strange and will result in incorrect amounts, which will cause problems for integrators.

- Example for usdc vault:

```
assetDecimal = 6
uint256 amount = vault.previewMint(shares) = 100 * 10**6
amount = Utils.changeDecimals(amount, 18, assetDecimal); = 0
res[j] = Utils.changeDecimals(amount, assetDecimal, 18) = 0
```

Recommendation: Store `amount` returned by mint preview as required collateral.

Berachain: Fixed in PR 481.

Spearbit: Not needed conversions that caused the issue were removed.

5.1.2 V2: Basketmode provides insufficient protection in severe depegs

Severity: Medium Risk

Context: HoneyFactory#mint

Description: If the pyth oracle reported stale price or the reported price deviate from the `lowerPegOffsets/upperPegOffsets`:

```
function isBasketModeEnabled() public view returns (bool basketMode) {
    if (forcedBasketMode) {
        return true;
    }

    for (uint256 i = 0; i < registeredAssets.length; i++) {
        if (!_isPegged(registeredAssets[i])) {
            return true;
        }
    }
    return false;
}
```

the basket mode is automatically activated, before the admin's intervention (admin update bad collateral), user can still mint honey with depegged asset.

Scenario:

- Suppose the Honey is backed by 65% USDT and 35% `DUMMY_TOKEN`.
- Oracle returns stale price.
- `DUMMY_TOKEN` depegs severely (suffers from an incident and can minted infinitely).
- Minter can offer 65% and 35% `DUMMY_TOKEN` to mint honey, the 35% `DUMMY_TOKEN` is worthless.
- Minter mints honey at 35% discount.

When basket mode is off, the minter can redeem honey with 100% good collateral.

```
function redeem(
    address asset,
    uint256 honeyAmount,
    address receiver
)
    external
    onlyRegisteredAsset(asset)
    whenNotPaused
    returns (uint256[] memory redeemed)
{
    bool basketMode = isBasketModeEnabled();
    if (!basketMode) {
        redeemed = new uint256[](1);
        redeemed[0] = _redeem(asset, honeyAmount, receiver);
        return redeemed;
    }
```

Recommendation: Do not mint Honey with a depegged asset in the basket mode or halt minting until an admin can intervene.

Berachain: The fix has been implemented on `honey/v2` branch.

Spearbit: Fixed.

5.1.3 V2: Single bad collateral block `HoneyFactory#mint` in basket mode forever

Severity: Medium Risk

Context: HoneyFactory#mint

Description: If there is a single bad collateral, the HoneyFactory#mint is blocked forever.

This is because as long as the asset depegs and the admin has to set the asset as bad collateral there is no way to remove a collateral.

Further, when minting in basket mode, the for loop at HoneyFactory.sol#L272 always loops through each registered asset.

```
for (uint256 i = 0; i < registeredAssets.length; i++) {
    amount = refAmount * weights[i] / 1e18;
    decimals = ERC20(registeredAssets[i]).decimals();
    amount = Utils.changeDecimals(amount, 18, decimals);

    honeyToMint += _mint(registeredAssets[i], amount, receiver, true);
}
```

Proof of Concept:

```
function test_mint_poc_one_bad_collateral_block_all_mint() external {

    uint256 _daiToMint = 100e18;

    uint256 mintedHoneys = (_daiToMint * daiMintRate) / 1e18;

    dai.approve(address(factory), _daiToMint);

    dai.approve(address(factory), _daiToMint);
    mintedHoneys = factory.mint(address(dai), _daiToMint, receiver);

    vm.prank(governance);

    bytes32 MANAGER_ROLE = keccak256("MANAGER_ROLE");

    factory.grantRole(MANAGER_ROLE, governance);

    vm.prank(governance);
    factory.setForcedBasketMode(true);

    vm.prank(governance);
    factory.setCollateralAssetStatus(address(usdt), true);

    dai.approve(address(factory), _daiToMint);
    mintedHoneys = factory.mint(address(dai), _daiToMint, receiver);

    // uint256 _usdtToMint = 10e6;
    // usdt.approve(address(factory), 10e6);
    // mintedHoneys = factory.mint(address(usdt), _usdtToMint, receiver);
}
```

The proof of concept reverts with the following error:

```
Ran 1 test suite in 157.23ms (9.56ms CPU time): 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/honey/HoneyFactory.t.sol:HoneyFactoryTest
[FAIL: AssetIsBadCollateral(0x2e234DAe75C793f67A35089C9d99245E1C58470b)]
```

Recommendation: Skip the mint inside the for loop if the registeredAssets[i] is bad collateral or if the amount is 0.

Berachain: Fixed in PR 481, because honey switches as emergency in basket mode.

Spearbit: In the current version, if basket mode is entered, the code not check if the asset is bad collateral, thus not blocking mint in the basket mode.

5.1.4 V2: Liquidated and 0 balance depegged asset causes basket mode

Severity: Medium Risk

Context: HoneyFactory.sol

Description: Basket mode switch is done with `isBasketModeEnabled()` function. It's currently designed to loop through all registered assets and detect if any of them have deviated.

```
for (uint256 i = 0; i < registeredAssets.length; i++) {
    if (!_isPegged(registeredAssets[i])) {
        return true;
    }
}
return false;
```

There are 2 points that this function doesn't consider:

- Basket mode should be switched off if bad collateral is fully liquidated.
- Basket mode should not turn on if balance of depegged asset is 0.

Recommendation: Described 2 points can be both solved if protocol ignores vaults with 0 balances. In case of bad collateral it will be not possible to mint and recapitalize with it as it's prohibited for bad collateral. As for the asset with 0 balance (newly added), basket mode will occur, once at least 1 share is minted with it. Protocol can consider introducing `minBalance` setting to ignore assets with small balances.

Berachain: We agree that the activation of the basket mode should be avoided when there are no bad collaterals.

Although we do not agree on ignoring depegged assets when not used as it allows a malicious user to mint honey with it (at a discount price) and promptly use such amount to redeem a pegged collateral asset.

Spearbit: Acknowledged.

5.1.5 V2: `HoneyFactory.mint()` may spend more funds than was expected during the basket mode switch

Severity: Medium Risk

Context: HoneyFactory.sol

Description: When `HoneyFactory` is in basket mode, then `amount` that user provides `mint()` function is taken as reference to calculate total honey amount to be minted. In case if not in basket mode, then `amount` itself is honey to be minted.

Protocol assumes that user knows when safety mode is switched on. But this may not be true and if user didn't expect that protocol is in basket mode, then more funds may be spent by user as more honey will be minted. Of course, user has to provide allowances for other tokens for this to happen.

Recommendation: Introduce `maxHoneyAmount` param that will be used as slippage protection.

Berachain: Acknowledged. We act on the user's interest on the frontend by providing the needed informations.

Should one interact directly on-chain, it would be his responsibility to check the status accordingly and approve the right amounts.

Should the basket mode be activated between the user action and the actual transaction execution, the user cannot spend more as the same amount is taken as grand total from all the collateral assets; he may even end up paying less due to a depegged asset.

Should the basket mode be disabled between the user action and the actual transaction execution, the operation will revert but without any major issue for him.

Spearbit: Acknowledged.

5.2 Low Risk

5.2.1 V2: `setPriceFeed` should be managed by `DEFAULT_ADMIN_ROLE`

Severity: Low Risk

Context: VaultAdmin.sol#L164

Description: There are two administrative roles in the Honey system:

- `DEFAULT_ADMIN_ROLE` responsible for upgrades, vault creation, and fee address management.
- `MANAGER_ROLE` setting bad asset status, pausing / unpausing.

The separation of roles makes sense in that one is slow and time locked while the other is less powerful but quicker to react. The manager may quickly respond to incidents by pausing while the admin may upgrade the contracts entirely.

Recommendation: Given that price feed alterations are akin to an upgrade, it is recommended to reserve that capability for the `DEFAULT_ADMIN_ROLE`.

The manager role is fine to set assets as bad assets, however, it may be better to reserve setting bad asset status to false for the `DEFAULT_ADMIN_ROLE`.

Berachain: Fixed in PR 481.

Spearbit: `setPriceFeed` function is now managed by `DEFAULT_ADMIN_ROLE`. Manager may still set a bad asset back to good.

5.2.2 V2: Asset settings made prior to vault creation are silently reset

Severity: Low Risk

Context: HoneyFactory.sol#L201-L203, HoneyFactory.sol#L211-L213

Description: Vault creation sets the `upper/lowerPegOffset` as well as the `relativeCap` for an `asset`. In scenarios where theses values are set prior to vault creation, the are silently overwritten by `HoneyFactory.createVault`.

A similar issue occurs when `referenceCollateral` is set prior to the first vault being created.

Recommendation: Add `onlyRegisteredAsset(asset)` modifier to `setDepegOffsets` and `setLiquidationRate`. Modify `HoneyFactory.createVault` to not override `referenceCollateral`:

```
- if (numRegisteredAssets() == 0) {
+ if (referenceCollateral == address(0) && numRegisteredAssets() == 0) {
      referenceCollateral = asset;
  }
```

Note: other findings suggest adding defaults for `mint/redeemRates` on vault creation, if those recommendations are accepted, those two functions should also have the modifier added.

Berachain: All those actions are designed to be called after the existence of a collateral vault, not before. We may consider to add the modifier shall the contract ends with enough free space.

Spearbit: Acknowledge.

5.2.3 V2: Mint and Redeem rates initialize to 0 allowing for 100% fee

Severity: Low Risk

Context: HoneyFactory.sol#L125-L136, HoneyFactory.sol#L139-L150, HoneyFactory.sol#L216-L225, HoneyFactory.sol#L495-L496, HoneyFactory.sol#L481

Description: `HoneyFactory.setRedeemRate` allows the `MANAGER_ROLE` to set what proportion of shares being redeemed remain after fees applied. The bounds are strictly enforced to be between 98% and 100% to restrict fees to a maximum of 2% total.

`HoneyFactory.createVault` does not initialize `redeemRates[asset]` making the default 0. This pushes the uninitialized fee to be 100% (significantly above the intended cap of 2%).

A similar issue occurs for `mintRates[asset]`.

Recommendation: Similar to `priceFeedMaxDelay`, `lowerPegOffsets`, and `upperPegOffsets`, setting redeemRates[asset]‘ to sound defaults (or no fees at all) would solve the unitialized value issue.

A less gas efficient but more strict safeguard may be added on mint and redeem to ensure fees are not in excess of the 2% cap.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.2.4 V2: Restrict `priceFeedMaxDelay`, `lowerPegOffsets` and `upperPegOffsets` to reasonable ranges

Severity: Low Risk

Context: HoneyFactory.sol#L173-L175

Description: The `MANAGER_ROLE` may set `priceFeedMaxDelay` to an arbitrarily large value. A similar unbounded amount is permitted when setting the peg offsets.

Recommendation: Restrict the setting of `priceFeedMaxDelay`, `lowerPegOffsets` and `upperPegOffsets` to be within an expected range. Scenarios requiring an extremely large delay, for example a halted feed, are more appropriate to be handled `DEFAULT_ADMIN_ROLE`.

Berachain: Fixed.

Spearbit: Fixed.

5.2.5 V2: Precision loss in `goodShares` refund logic in `HoneyFactory.liquidation`

Severity: Low Risk

Context: HoneyFactory#liquidation

Description: In liquidation, if the `badAmount` exceed the `badShares`, the code refunds the `goodShares`.

```
uint256 goodShares = vaults[goodCollateral].deposit(goodAmount, address(this));

uint256 priceBad = _getPrice(badCollateral);
uint256 priceGood = _getPrice(goodCollateral);
badAmount = (goodShares * priceGood / priceBad) * (1e18 + liquidationRates[badCollateral]) / 1e18;

uint256 badShares = vaults[badCollateral].balanceOf(address(this));
if (badAmount > badShares) {
    uint256 goodSharesAdjusted = badShares * goodShares / badAmount;
    vaults[goodCollateral].redeem(goodShares - goodSharesAdjusted, msg.sender, address(this));
    badAmount = badShares;
}
```

But if the `badAmount` is a large number, just because `priceBad` is too small when the asset depeg (depeg → asset price drop to dust):

```
uint256 goodSharesAdjusted = badShares * goodShares / badAmount;
```

the pgoodSharesAdjusted‘ is rounded down and can be round down to 0 in the worst case,

```
vaults[goodCollateral].redeem(goodShares, msg.sender, address(this));
```

then all good shares get refunded and user take bad collateral for free without adding good collateral.

Recommendation: Remove the good share refund logic, the liquidator already reward with the liquidation bonus.

```
uint256 goodShares = vaults[goodCollateral].deposit(goodAmount, address(this));

uint256 priceBad = _getPrice(badCollateral);
uint256 priceGood = _getPrice(goodCollateral);
badAmount = (goodShares * priceGood / priceBad) * (1e18 + liquidationRates[badCollateral]) / 1e18;

uint256 badShares = vaults[badCollateral].balanceOf(address(this));
if (badAmount > badShares) {
    badAmount = badShares;
}
```

It is also recommended to add a parameter `minAmountOut` to serve as a slippage protection for liquidator.

Berachain: Fixed.

Spearbit: Fixed.

5.2.6 V2: Frontrunning `HoneyFactory#recapitalize` via donating can DoS `HoneyFactory#mint` if user approve exact token amount

Severity: Low Risk

Context: HoneyFactory#recapitalize

Description: If the basket mode is activated, the user has to provide asset in proportion to mint honey. The asset in proportion refers as the "*weight*". Each asset required to mint honey is computed from the weight as well.

The problem is that the weight can be changed by calling `HoneyFactory#recapitalize` donation.

1. Alice want to mint honey when the basket mode is activated, user check that he needs to approve 70 USDT and 30 DAI to mint honey.
2. Bob frontruns the mint and donate a dust amount of USDT to change the weight of USDT and DAI.
3. Alice's mint transaction reverts because the USDT required to mint honey is more than 70 USDT.

Proof of Concept:

```
function test_mint_poc_recap_dos() external {

    uint256 _daiToMint = 100e18;
    uint256 mintedHoneys = (_daiToMint * daiMintRate) / 1e18;

    dai.approve(address(factory), _daiToMint);

    dai.approve(address(factory), _daiToMint);
    mintedHoneys = factory.mint(address(dai), _daiToMint, receiver);

    uint256 _usdtToMint = 10e6;
    usdt.approve(address(factory), 10e6);
    mintedHoneys = factory.mint(address(usdt), _usdtToMint, receiver);

    vm.prank(governance);

    bytes32 MANAGER_ROLE = keccak256("MANAGER_ROLE");

    factory.grantRole(MANAGER_ROLE, governance);

    vm.prank(governance);
    factory.setForcedBasketMode(true);

    // uint256 recap_amount = 5 wei;

    // usdt.mint(address(this), recap_amount);
    // usdt.approve(address(factory), recap_amount);
    // factory.recapitalize(address(usdt), recap_amount);

    usdt.approve(address(factory), 10000000);

    dai.approve(address(factory), _daiToMint);
    mintedHoneys = factory.mint(address(dai), _daiToMint, receiver);

}
```

The proof of concept works fine, but if we uncomment the recap code → stimulating that someone recap to change the weight, the transaction reverts because the previous approval is not sufficient.

Recommendation: `HoneyFactory#recapitalize` can change the asset weight directly. It is recommended to add access control to this function.

Berachain: It does not really seems an issue: one is gifting money and the other has to adjust his allowances.

In order to mitigate it, we have chosen to set a minimum amount that can be gifted in order to make it economically infeasible and a target balance to avoid exceeding the needed liquidity.

Fixed in PR 481.

Spearbit: As recapitalization requires a minimum amount to be sent, then it's indeed not economically feasible for a griefer. Fixed; the protocol set a minimum amount that can be donated in order to make it economically infeasible and a target balance to avoid exceeding the needed liquidity.

5.2.7 V2: `HoneyFactory#mint` contain division by zero error when basket mode is activated.

Severity: Low Risk

Context: HoneyFactory.sol#mint

Description: Suppose the honey is backed by USDT and DAI. In basket mode, if an asset does not have any token balance (honey is backed by 100% DAI and 0 USDT):

```
uint256[] memory weights = _getWeights(false);
  // Here the assumption is that the callers knows about the basket mode and
  // has already approved all the needed (and previewed) amounts.
  // As we cannot trust the caller to have provided the right asset/amount tuples
  // without changing their distribution or without tricking it (e.g. by repeating some assets),
  // we take one of those amount as reference and compute the others accordingly.
  uint8 decimals = ERC20(asset).decimals();
  uint256 refAmount = Utils.changeDecimals(amount, decimals, 18);

  // console.log("value:", weights[_lookupRegistrationIndex(asset)]);
  refAmount = refAmount * 1e18 / weights[_lookupRegistrationIndex(asset)];

  for (uint256 i = 0; i < registeredAssets.length; i++) {
      amount = refAmount * weights[i] / 1e18;
      decimals = ERC20(registeredAssets[i]).decimals();
      amount = Utils.changeDecimals(amount, 18, decimals);
      console.log("amount:", amount);
      console.log("asset:", ERC20(registeredAssets[i]).symbol());
      honeyToMint += _mint(registeredAssets[i], amount, receiver, true);
  }
```

Minting with USDT in basket mode will revert in division by zero in HoneyFactory.sol#L270.

```
refAmount = refAmount * 1e18 / weights[_lookupRegistrationIndex(asset)];
```

the weights[_lookupRegistrationIndex(asset)] is 0. Can add the coded proof of concept to `HoneyFactory.t.sol`.

```
function test_mint_poc_division_by_zero() external {
    uint256 _daiToMint = 100e18;
    uint256 mintedHoneys = (_daiToMint * daiMintRate) / 1e18;
    dai.approve(address(factory), _daiToMint);

    vm.prank(governance);

    bytes32 MANAGER_ROLE = keccak256("MANAGER_ROLE");

    factory.grantRole(MANAGER_ROLE, governance);

    vm.prank(governance);
    factory.setForcedBasketMode(true);

    dai.approve(address(factory), _daiToMint);
    mintedHoneys = factory.mint(address(dai), _daiToMint, receiver);

}
```

we can run with the command:

```
forge test -vv --match-test "test_min_poc_division_by_zero"
```

output:

```
Failing tests:
Encountered 1 failing test in test/honey/HoneyFactory.t.sol:HoneyFactoryTest
[FAIL: panic: division or modulo by zero (0x12)]
```

Recommendation: Handle the division by zero case gracefully.

Berachain: Fixed in PR 481.

Spearbit: The code now reverts with an explicit error message.

5.2.8 V2: Caps are not checked after redemption in normal mode

Severity: Low Risk

Context: HoneyFactory.sol

Description: After each honey mint, protocol checks if the global or relative cap is not breached with the intent to collect funds in a determined proportion.

However, those caps are not checked after the redemption in usual mode. Because of that, it's possible to change the relative weight of an asset. It can be done with a whale redemption or just a large amount of usual redemptions.

Recommendation: Check both global and relative caps after redemption.

Berachain: Addressed in commit d57beacc. We have analyzed it, and we have deemed necessary to check the global cap and, when the asset is the relative one, also check the relative cap of all the other collateral assets.

Spearbit: There is relative and global cap check for redemptions now.

5.2.9 Consider introduce mint / redeem timelock to further mitigate cross-asset arbitrage

Severity: Low Risk

Context: HoneyFactory.sol

Description: Protocol is aware that in case when there are multiple asset, user can mint honey with asset A and redeem asset B.

> An user can always enter with one asset and exit with the other.
>
> As a result, any depeg creates an arbitrage opportunity:
>
> - buy depegged asset
> - mint at 1:1 honey
> - redeem not-depegged assets

Recommendation: While adjusting minting and redeeming fee can partially reducer user's incentive to cross-asset arbitrage.

It is recommended to introduce mint / redeem timelock to further mitigate the issue. Both adding fee and introduce timelock can make such arbitrage non-profitable.

Berachain: Acknowledged. We expect honey to have a big user base. As a result a time lock could make many transactions revert, resulting in a UX not acceptable. Furthermore, the time lock must be short and therefore the abribtrage can still be executed relatively easily by a sophisticated actor.

Spearbit: Acknowledged.

5.3 Gas Optimization

5.3.1 V2: Do not `_mint` or `_redeem` when amount is 0

Severity: Gas Optimization

Context: HoneyFactory.sol#L296, HoneyFactory.sol#L305, HoneyFactory.sol#L257, HoneyFactory.sol#L277

Description: Minting and redeeming allow for passing in 0 for `amount/honeyAmount`. Further, in basket mode, it is possible for the calculated amounts to round down to 0, even when weights are non zero.

Recommendation: In basket mode, skip the mint/redeem internal function call if amounts are 0. Skip when weights are 0 for the asset as well.

Consider reverting prior to minting or redeeming 0 amounts when 0 passed in as a param. If adopted this would also apply to liquidate and recapitalize.

Berachain: The fix has been implemented on `honey/v2`.

Spearbit: Fixed.

5.3.2 V2: Liquidate should fail when there are 0 `badShares`

Severity: Gas Optimization

Context: HoneyFactory.sol#L342

Description: Similar to slippage protection, if there are 0 `badShares` the transaction should revert to defend against both user error and having been front run.

Adding this check early in the function call will save gas involved in the `goodShares - goodSharesAdjusted` refund logic.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.3.3 V2: Returning weights as a numerator and denominator can eliminate one loop

Severity: Gas Optimization

Context: HoneyFactory.sol#L520-L522

Description: Everywhere `_getWeights` is called ends up looping over them. Performing the division by `sum` in the later loop, and not withing `_getWeights` would cut looping over weights down from 3x to 2x.

This would require an edit to `_getWeights` as well as to the functions calling `_getWeights`.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.3.4 V2: Store `registrationIndex` to save on gas

Severity: Gas Optimization

Context: VaultAdmin.sol#L301

Description: Searching for the registration index but looping over assets incurs more `sloads` than is necessary. Storing the index on vault creation would mean only a single `sload` is required.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.3.5 Consider immutables for gas efficiency

Severity: Gas Optimization

Context: Honey.sol#L20, HoneyFactory.sol#L26

Description: Some variables are known, or knowable, at deploy time and can be made immutable.

Recommendation: The two way referencing between `Honey` and `HoneyFactory` means both addresses can't be precalcaulted `honey` and `factory` are made immutable. A workaround is to have the factory deploy `Honey` in the constructor and store the address as immutable. Or to calculate both proxy addresses prior to deploying their respective implementations.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.4 Informational

5.4.1 V2: Depegged assets received on redeem even when Honey is fully recapitalized

Severity: Informational

Context: HoneyFactory.sol#L360-L378

Description: Recapitalization allows a partially depegged basket to be brought back to 100% backing. The happy path for recapitalization is to:

1. Flag the bad collateral.
2. Liquidate the bad collateral (swap to remove from system).
3. Recapitalize (donate good collateral to bring the backing up to 100%).

In the event step 3 is performed prior to step 2 concluding, on redeem during basketmode, bad collateral will be given even though the system is full backed.

Recommendation: At minimum document the implications of redeeming when there is bad collateral in the system.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.4.2 V2: Move test functions to a test only contract

Severity: Informational

Context: HoneyFactory.sol#L101-L118

Description: `HoneyFactory.initializeBasketMode` is noted to be for test purposes only by the development team. To avoid added surface area in the production contract, it's recommended to move this function to a test only contract that extends `HoneyFactory`.

Berachain: Acknowledged (removed on `honey/v2` branch).

Spearbit: Acknowledged.

5.4.3 V2: Address pending TODOs and TBDs

Severity: Informational

Context: VaultAdmin.sol#L165

Description: Setting price feed includes a comment `TBD: check that the feed value is about 1.`

Recommendation: Some validation on the price feed is recommended. Caution that if the current price is too strictly bound, it would prevent updating feeds in the case of a sustained depeg and may prevent liquidations.

Berachain: Fixed.

Spearbit: Fixed.

5.4.4 V2: Consider adding more testing for liquidation

Severity: Informational

Context: HoneyFactory

Description: There is lack of testing for liquidation.

Recommendation: Consider add more test case for liquidation.

Berachain: Acknowledged.

Spearbit: Acknowledged.

5.4.5 V2: State update function missing event emission

Severity: Informational

Context: HoneyFactory.sol

Description: State update functions do not emit events.

Recommendation: Emit events in the functions below:

- HoneyFactory.sol#L173: `setMaxFeedDelay`.
- HoneyFactory.sol#L201: `setLiquidationRate`.
- HoneyFactory.sol#L206: `setGlobalCap`.
- HoneyFactory.sol#L211: `setRelativeCap`.

VaultAdmin.createVault currently emits and event but would benefit from adding the feed to the event, or to emit `PriceFeedChanged`.

Berachain: Fixed.

Spearbit: Fixed.

5.4.6 Consider adding storage gap in upgradeable contracts

Severity: Informational

Context: HoneyFactory.sol#L16, Honey.sol#L13, CollateralVault.sol#L13

Description: Consider adding storage gap in upgradeable contracts. As these 3 contracts are not extended, the opportunity for error is low.

Berachain: Rejected.

Spearbit: Acknowledged.

5.4.7 Validate all token addresses have code or make use of OZ's `safeTransfer`

Severity: Informational

Context: HoneyFactory.sol#L138-L140, *(out of scope files not noted but some do make use of the same pattern)*.

Description: When using Solady `SafeTransferLib` and not validating a token has code deployed to its address at the time of transfer (contrasting with the OZ `safeTransfer` behavior), it is important to validate elsewhere that tokens have code deployed at their address.

While this issue is information in nature, when combined with other aspects of the code base, vulnerabilities can surface.

For the current implementation of Honey, collateral vaults will revert on `initialize` if there is no token at the `_asset` address.

Both OpenZeppelin's and the later Solady releases include an address check if no data returned and would remediate the potential issue.

Berachain: Upgraded to a later Solady version in PR 472.

Spearbit: Fixed.

5.4.8 Consider initialization of `PausableUpgradeable` inside `CollateralVault` and `VaultAdmin` contracts

Severity: Informational

Context: CollateralVault.sol, VaultAdmin.sol

Description: `CollateralVault` and `VaultAdmin` contracts extend `PausableUpgradeable`, which has `__Pausable_init` function that is designed to be called on child initialization. Currently, those contracts don't call it.

Recommendation: Consider adding initialization of `PausableUpgradeable` inside `CollateralVault` and `VaultAdmin` contracts.

Berachain: Fixed.

Spearbit: Fixed.

5.4.9 Consider making `mintRates` and `redeemRates` public

Severity: Informational

Context: HoneyFactory.sol

Description: In honey factory, the minRates and redeemRates are internal variables:

```
/// @notice Mint rate of Honey for each asset, 60.18-decimal fixed-point number representation
mapping(address asset => uint256 rate) internal mintRates;
/// @notice Redemption rate of Honey for each asset, 60.18-decimal fixed-point number representation
mapping(address asset => uint256 rate) internal redeemRates;
```

There is no way for user to query the asset mint and redeem rate directly.

Recommendation:

```
/// @notice Mint rate of Honey for each asset, 60.18-decimal fixed-point number representation
mapping(address asset => uint256 rate) public mintRates;
/// @notice Redemption rate of Honey for each asset, 60.18-decimal fixed-point number representation
mapping(address asset => uint256 rate) public redeemRates;
```

Berachain: Fixed in PR 481.

Spearbit: Fixed.

5.4.10 Consider reverting transaction early with explicit check and revert reason when a specific vault is paused

Severity: Informational

Context: HoneyFactory

Description: HoneyFactory#mint function needs to call vault#deposit:

```
uint256 shares = vault.deposit(amount, address(this));
```

If the contract is paused, the `HoneyFactory#mint` will revert because of the `WhenNotPaused` modifier. If the specific vault is paused, the vault.deposit will revert as well.

In the current code, there is no way to tell if the mint revert because a specific vault is paused or because the factory contract is paused.

Recommendation:

```
/// @inheritdoc IHoneyFactory
function mint(
    address asset,
    uint256 amount,
    address receiver
)
    external
    onlyRegisteredAsset(asset)
    onlyGoodCollateralAsset(asset)
    whenNotPaused
    checkInvariants(asset)
    returns (uint256)
{
    ERC4626 vault = vaults[asset];
    if (vault.paused()) {
            revert("vault is paused");
    }
```

Berachain: Fixed.

Spearbit: Fixed.