Berachain Honey #2

Defensive Fuzzing Report

Feb. 4, 2025

Prepared By:

0xScourgedev | Lead Fuzzing Specialist
0xscourgedev@perimetersec.io

Rappie | Lead Fuzzing Specialist
rappie@perimetersec.io

Table of Contents

About Perimeter

Perimeter's mission is to deliver the highest quality fuzzing services to protocols by uniting the world's foremost fuzzing specialists. We possess extensive expertise in fuzzing a diverse range of protocols, from smaller, niche protocols to some of the largest and most complex in DeFi.

In order to deliver on our mission, we have developed the most advanced scaffolding and libraries, enabling us to create highly sophisticated fuzzing suites tailored to meet the unique challenges of each protocol.

Learn more about us at perimetersec.io.

Risk Classification

The severity of security issues identified during the security review is classified according to the table below.

A. Critical findings are highly likely to be exploited with severe impact on the protocol and require immediate attention.

B. High findings are very likely to occur, easy to exploit, or difficult but highly incentivized, and should be resolved as quickly as possible.

C. Medium findings are possible in certain circumstances or when incentivized, with a moderate likelihood of occurring, and should be addressed.

D. Low findings involve rare circumstances to exploit or offer little to no incentives, though addressing them is still recommended.

E. Informational issues represent improvements that do not impact the project's overall security but are worth considering.

Severity Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Services Provided

Perimeter has successfully delivered a comprehensive suite of services that include:

- **Fuzzing Suite Development:** Design and implement a stateful fuzzing suite using Echidna. This suite will be tailor-made for the protocol and contracts in scope. The completed fuzzing suite can later be integrated into the testing suite to serve long-term security needs.
- **Findings Reporting:** We provided thorough documentation and reporting of all findings identified throughout the engagement.
- **Invariant Testing Assurance:** Guarantee that each invariant implemented will be tested no fewer than 50,000,000 instances, ensuring thorough validation and reliability.
- **Proof-of-Concept Development:** Develop a corresponding Proof-of-Concept (PoC) for each finding and assertion/property counterexample identified, to demonstrate potential vulnerabilities and their implications.
- **Comprehensive Final Report:** Create a detailed final report that will include all findings, along with their corresponding PoCs. This report will also detail the invariants tested, their run status, and the number of runs, providing a comprehensive overview of the engagement's outcomes.

Files in Scope

The engagement will be focused on the files listed below, acquired from commit 921170edcfdbda6e326092869870d0a47489abec.

```
src
├── extras
│   └── IPriceOracle.sol
├── honey
│   ├── CollateralVault.sol
│   ├── HoneyFactoryReader.sol
│   ├── HoneyFactory.sol
│   ├── Honey.sol
│   ├── IHoneyErrors.sol
│   ├── IHoneyFactory.sol
│   └── VaultAdmin.sol
└── libraries
    └── Utils.sol
```

Files Out of Scope

Files outside the scope were not directly considered in achieving the target. However, since many of these files are utilized by those within the scope, a significant portion was indirectly covered.

Methodology

The primary objectives of this engagement, in continuation of the first Honey engagement, were to update the fuzzing suite to align with the updated Honey codebase, implement additional invariants, and ensure that issues identified in the initial engagement did not compromise new or existing properties.

Due to changes in the Honey codebase, many existing invariants no longer conformed to the updated specifications. As a result, these invariants were either updated or deprecated, and new invariants reflecting the revised specifications were added to the fuzzing suite.

Some invariants required tolerances to account for rounding errors, which necessitated thorough investigations. Using Echidna's optimization mode, we identified the maximum rounding errors and verified that they remained within acceptable tolerance levels. This approach ensured accurate and realistic testing while recognizing the inevitability of rounding errors.

Significant effort was dedicated to investigating the root cause of an underflow that caused multiple invariants to fail. The issue was ultimately traced to a deeply hidden issue in the `_redeem` function. The findings from these investigations are documented under issue MEDIUM-01.

To improve the efficiency of the fuzzing process, bulk actions were introduced, enabling the fuzzer to navigate the large search space of the updated fuzzing suite more effectively.

Issues

At Perimeter, our objective is to thoroughly investigate and uncover critical vulnerabilities through fuzzing and reveal issues often overlooked in manual reviews. Any lower-severity findings are incidental to this core focus.

Severity	Count	Fixed	Partially Resolved	Acknowledged
Critical	0	0	0	0
High	1	0	1	0
Medium	1	1	0	0
Low	0	0	0	0
Informational	1	1	0	0
Total	**3**	**2**	**1**	**0**

HIGH-01: Front-run of Lowering of Global Cap Leads to Denial of Service

Description

The `setGlobalCap` function allows modification of the global cap without validation, enabling it to be set to values that deviate from the protocol's intended constraints. This lack of validation can lead to scenarios where the current state is no longer considered globally capped. Consequently, the `_isCappedGlobal` function returns `false`, resulting in a denial of service for critical actions such as minting and redeeming.

A malicious actor can exploit this vulnerability by front-running a reduction in the global cap. They can manipulate the weight of a collateral asset to fall between the new lower cap and the previous higher cap.

The manipulation is achieved in two primary ways:

- **Minting**: The malicious actor mints Honey and provides the target asset as collateral. This increases the target asset's weight as its share grows while others remain constant.
- **Redeeming**: The malicious actor redeems Honey for assets other than the target collateral. This decreases the shares of other assets, increasing the target asset's relative weight.

Since Honey is a stablecoin backed 1:1 by other stablecoins, a malicious actor can leverage flash loans to significantly lower the capital required to execute the attack. The two primary attack flows using a flash loan are as follows:

- **Minting**:
    1. The attacker takes out a flash loan of the target asset.
    2. They use the target asset to mint Honey at a 1:1 ratio.
    3. The attacker swaps the minted Honey back into the target asset.
    4. Finally, the flash loan is repaid.
- **Redeeming**:
    1. The attacker takes out a flash loan of Honey.
    2. They redeem the Honey for a stablecoin collateral asset at a 1:1 ratio.
    3. Attacker swaps the redeemed stablecoin back into Honey.
    4. Finally, the flash loan is repaid.

By using flash loans, the cost of executing this attack is limited to the minting or redeeming fees charged by the `HoneyFactory`, plus the associated swap fees.

Recommendation

Allow actions that reduce the weight of overcapped collaterals. Additionally, use mitigation strategies against this attack. For example by lowering the global cap incrementally with monitoring for this attack and using private mempools.

Remediation Findings

A. **Perimeter:** The effects of setting the cap below the current weight is mitigated by being able to mint the other assets to reduce the weight of the asset that is above the global cap. However the other way of reducing the weight of the "overcapped" asset is by redeeming for it. However redeeming is still blocked for every asset due to that one asset being above the global cap. Therefore it would require even more capital to reduce the weight of the "overcapped" asset, while reducing the weight of the "overcapped" asset via redeem requires none.

Here's the POC for the redeem scenario:

```
function test_exploit() public {
    honeyFactory.mint(address(token1), 4e18, address(this), false);
    honeyFactory.mint(address(token2), 3e18, address(this), false);
    honeyFactory.mint(address(token3), 3e18, address(this), false);
    emit DebugWeights(honeyFactory.getWeights()); // Weights:
[400000000000000000 [4e17], 300000000000000000 [3e17], 300000000000000000
[3e17]]

    // Set the starting global cap to 50%
    HoneyFactoryFuzzWrapper(honeyFactory).setGlobalCap(0.5e18); // 50%

    // Frontrunning transaction to mint more token2 to be between 40% and 50%
of the total weight
    honeyFactory.mint(address(token2), 3e18, address(this), false);
    emit DebugWeights(honeyFactory.getWeights()); // Weights:
[307692307692307692 [3.076e17], 461538461538461538 [4.615e17],
230769230769230769 [2.307e17]]
    // Reduce the global cap to 40%
    HoneyFactoryFuzzWrapper(honeyFactory).setGlobalCap(0.4e18); // 40%

    // Try to mint for an asset that's below the global cap (and it's the
reference asset) to reduce the weight of token2
    // This now passes with the new code, and successfully reduces the weight
of the "over-capped" token2
    honeyFactory.mint(address(token1), 1e18, address(this), false);
    emit DebugWeights(honeyFactory.getWeights()); // Weights:
[357142857142857142 [3.571e17], 428571428571428571 [4.285e17],
214285714285714285 [2.142e17]]
```

```
    // Try to redeem for token2 to get below the global cap
    // This still reverts due to exceeding the global cap
    honeyFactory.redeem(address(token2), 1e17, address(this), false);
}
```

Berachain: Yeah, we thought a bit and we think that adding the following code within _isCappedGlobal should do the job:

```
if (!isMint && registeredAssets[i] == collateralAsset) {
       continue;
}
```

B. **Perimeter:** There is a separate issue now with the fixes https://github.com/berachain/contracts-monorepo/pull/566/files#diff-dc6559e6897d75471be317faec6fe33dacc22a0033f297a322df1b78faafddc1R380 and https://github.com/berachain/contracts-monorepo/pull/566/files#diff-dc6559e6897d75471be317faec6fe33dacc22a0033f297a322df1b78faafddc1R443 are using the `address(0)` as the collateral address, making this check not work

Berachain: Fixed

C. **Perimeter:** If the attacker pushes the cap of two tokens above the global cap, the minting will not be DoS'd, however redeeming will.

Here's a POC showing the issue:

```
function test_exploit() public {
    honeyFactory.mint(address(token1), 1e18, address(this), false);
    honeyFactory.mint(address(token2), 1e18, address(this), false);
    honeyFactory.mint(address(token3), 1e18, address(this), false);
    honeyFactory.mint(address(token4), 1e18, address(this), false);
    emit DebugWeights(honeyFactory.getWeights()); // Weights:
[250000000000000000 [2.5e17], 250000000000000000 [2.5e17], 250000000000000000
[2.5e17], 250000000000000000 [2.5e17]]
    // Set the starting global cap to 50%
    HoneyFactoryFuzzWrapper(honeyFactory).setGlobalCap(0.5e18); // 50%

    // Frontrunning transaction to mint more token2 and token3 to be between
30% and 50% of the total weight
    honeyFactory.mint(address(token2), 1e18, address(this), false);
    honeyFactory.mint(address(token3), 1e18, address(this), false);
    emit DebugWeights(honeyFactory.getWeights()); // Weights:
[166666666666666666 [1.666e17], 333333333333333333 [3.333e17],
333333333333333333 [3.333e17], 166666666666666666 [1.666e17]])
```

```
    // Reduce the global cap to 30%
    HoneyFactoryFuzzWrapper(honeyFactory).setGlobalCap(0.3e18); // 30%

    // Try to mint for an asset that's below the global cap (and it's the
reference asset) to reduce the weight of token2 and token3
    // This passes, since it only checks the weight of the minted asset
    // honeyFactory.mint(address(token1), 1e18, address(this), false);
    // emit DebugWeights(honeyFactory.getWeights()); // Weights:
[285714285714285714 [2.857e17], 285714285714285714 [2.857e17],
285714285714285714 [2.857e17], 142857142857142857 [1.428e17]])

    // Try to redeem for token2 to get below the global cap
    // This fails because it checks the weight of every other asset that is
not the one being redeemed
    // Since there are two assets that are over cap, redeeming one overcapped
asset will increase the weight of the other overcapped asset
    honeyFactory.redeem(address(token2), 1e18, address(this), false);
}
```

Response

In order to protect mint and burn functionalities, we chose to let `setGlobalCap` revert when it attempts to set a limit too low for the current state

To manage reverts a multitude of Ops strategies can be applied:

- Setting caps not close to the amount of collateral in a vault.
- If it is needed to set the cap close to the amount of collateral, first pause the collateral vault and then change caps.
- If it is needed to set the cap below the amount of collateral, first use `liquidate` function to reduce the amount in the vault and then change caps.
- Regarding the usage of flash loans to force a revert, there's also another implicit mitigation given by the fact that we can attempt multiple reductions of cap, thus making the attack less and less economically sustainable.
- In addition we are evaluating the usage of a private mempool, when applicable.

Proof of Concept

```
function test_exploit() public {
  honeyFactory.mint(address(token1), 4e18, address(this), false);
  honeyFactory.mint(address(token2), 3e18, address(this), false);
  honeyFactory.mint(address(token3), 3e18, address(this), false);
  emit DebugWeights(honeyFactory.getWeights()); // Weights: [400000000000000000
[4e17], 300000000000000000 [3e17], 300000000000000000 [3e17]]

  // Set the starting global cap to 50%
  HoneyFactoryFuzzWrapper(honeyFactory).setGlobalCap(0.5e18); // 50%

  // Frontrunning transaction to mint more token2 to be between 40% and 50% of the
total weight
  honeyFactory.mint(address(token2), 3e18, address(this), false);
  emit DebugWeights(honeyFactory.getWeights()); // Weights: [307692307692307692
[3.076e17], 461538461538461538 [4.615e17], 230769230769230769 [2.307e17]]

  // Reduce the global cap to 40%
  HoneyFactoryFuzzWrapper(honeyFactory).setGlobalCap(0.4e18); // 40%

  // Try to mint for an asset that's below the global cap (and it's the reference
asset) to reduce the weight of token2
  // This reverts due to exceeding the global cap
  honeyFactory.mint(address(token1), 1, address(this), false);

  // Try to redeem for token2 to get below the global cap
  // This reverts due to exceeding the global cap
  honeyFactory.redeem(address(token2), 4e18, address(this), false);
}
```

MEDIUM-01: Potential Denial of Service on Fee Withdrawals Due to Missing Redemption Validation

Description

The `redeem` function relies on implicit downstream reverts to enforce limits on the amount of honey redeemed rather than performing explicit checks. This reliance creates gaps in validation, allowing potential edge cases to bypass safeguards under specific conditions.

In the case where basket mode is **disabled**, two validation paths depend on downstream reverts but can be bypassed:

1. `_isCappedRelative` does not trigger when non-reference collateral is redeemed.
2. `_isCappedGlobal` does not trigger when bad collateral is redeemed.

When basket mode is **enabled**, redemptions are balanced across all vaults, making checks unnecessary. However, liquidation premiums distort the ratio of honey to vault shares, creating opportunities for excessive collateral withdrawals until the system is recapitalized.

No collateral can be stolen, as users can redeem honey 1:1 against any collateral asset. However, the issue arises from the way fees are managed. Fees are calculated and stored individually for each vault but not tied directly to the vault's remaining shares.

If excessive redemptions are performed on a single vault, the remaining shares can fall below the collectible fees. Since fee withdrawals always attempt to withdraw the full amount of collectible fees, this results in a revert when the shares are insufficient to cover the outstanding fees.

In addition to causing a denial of service on fee withdrawals, this issue introduces state inconsistencies that can have broader downstream effects, including but not limited to:

- Underflows in the `_getSharesWithoutFees` function
- Unintended reverts in functions such as `liquidate`, `recapitalize`, and `getWeights`

Recommendation

Introduce a validation mechanism to ensure that the amount of honey being redeemed does not exceed the total supply of the vault minus the collectible fees. This safeguard will guarantee that sufficient shares remain in the vault to cover outstanding fees, preventing scenarios where fee withdrawals result in reverts due to insufficient shares.

Although the current impact focuses on reverts during fee withdrawals, the absence of this check is potentially much riskier than the current impact describes. While no collateral can be stolen under normal circumstances, the lack of explicit validation introduces systemic vulnerabilities that could lead to unintended behavior or unforeseen edge cases.

Response

Fixed.

Proof of Concept

```
function test_withdrawFeeDenialOfService() public {
  // Set PoL fee rate to a low amount to have fees go to feeReceiver
  honeyFactory.setPOLFeeCollectorFeeRate(1e18 / 10);

  // Mint initial tokens
  honeyFactory.mint(address(token1), 10e18, address(this), false);
  honeyFactory.mint(address(token2), 10e18, address(this), false);

  // Accumulate fees by repeatedly minting and redeeming
  for (uint256 i = 0; i < 100; i++) {
      honeyFactory.mint(address(token2), 1e18, address(this), false);
      honeyFactory.redeem(address(token2), 1e18, address(this), false);
  }

  // Mark Token 2 as bad collateral to bypass checks
  honeyFactory.setCollateralAssetStatus(address(token2), true);

  // Redeem the the entire supply of shares for token 2
  honeyFactory.redeem(address(token2), vault2.totalSupply(), address(this), false);

    // Debug prints
  uint256 fees = honeyFactory.collectedFees(FEE_RECEIVER, address(token2));
  uint256 supply = vault2.totalSupply();
  console.log("Collected fees", fees);        // 3988440000000000001
  console.log("Vault2 total supply", supply); //  2084400000000000000

  // Reverts with `RedeemMoreThanMax()`
  honeyFactory.withdrawFee(address(token2), FEE_RECEIVER);
}
```

INFO-01: Missing Error Signature Comment for `UnexpectedBasketModeStatus()`

Description

The `UnexpectedBasketModeStatus` error lacks an accompanying comment explicitly describing its signature, which is inconsistent with all other errors in the `IHoneyErrors.sol` file.

Recommendation

Add a comment detailing the signature of `UnexpectedBasketModeStatus` to maintain consistency and improve code documentation.

Response

Fixed.

Invariants

We created many tests to verify the correctness of **53** invariants described in the table below. During the execution phase, these invariants were assessed for a total of **1,110,000,000+** calls.

The table below lists all invariants that are part of this engagement.

Invariant	Description	Tested	Passed	# Runs
MINT-01	Minting does not unexpectedly revert	✅	❌	-
MINT-02	When basket mode is not enabled, minting decreases the user's asset balance by the inputted amount	✅	✅	1.11B+
MINT-03	If the inputted amount is not 0 with normalized decimals, the mint rate is one hundred percent and basket mode for mint is disabled, then the honey balance of the receiver strictly increases after a successful mint	✅	✅	1.11B+
MINT-04	The receiver's honey balance increases by the returned honeyToMint amount after a successful mint	✅	✅	1.11B+
MINT-05	When basket mode is disabled, the difference in the recipient's honey balance is less than or equal to the difference in the selected vault's total supply	✅	✅	1.11B+
MINT-07	If the mint rates for the asset is 100%, then the the collected fees for the fee receiver does not change after a successful mint	✅	✅	1.11B+
MINT-08	If the mint rates for the asset is 100%, then the collected fees for the POL fee collector does not change after a successful mint	✅	✅	1.11B+

Invariant	Description	Tested	Passed	# Runs
MINT-09	If the mint rates for the asset is less than 100%, the inputted amount is not 0 and the preview deposit amount is not 0, then the difference in total supply of the selected asset vault be strictly less than the previewed deposit amount and/or the collectedFees is strictly increasing	✅	✅	1.11B+
MINT-10	If the mint rates for the asset is 100%, the difference in honey total supply is equal to the difference in the receiver's honey balance after a successful mint	✅	✅	1.11B+
MINT-11	When basket mode is enabled, if the honey balance of the user increased, then the asset balance of at least one registered asset is strictly decreasing	✅	✅	1.11B+
REDEEM-01	Redeeming does not unexpectedly revert	✅	❌	-
REDEEM-02	The honey balance of the caller decreases by exactly the inputted amount after a successful redemption	✅	✅	1.11B+
REDEEM-03	If the inputted amount is not 0, the redeem rate is 100% and basket mode for redeem is disabled, the balance of asset for the receiver strictly increases after a successful redemption	✅	✅	1.11B+
REDEEM-04	When basket mode is disabled and the redeem rate is 100%, the difference in the user's honey balance is less than or equal to the difference in the selected vault's total supply after a successful redeem	✅	✅	1.11B+
REDEEM-06	If the redeem rates for the asset is 100%, then the the collected fees for the fee receiver does not change after a successful redeem	✅	✅	1.11B+
REDEEM-07	If the redeem rates for the asset is 100%, then the collected fees for the POL fee collector does not change after a successful redeem	✅	✅	1.11B+
REDEEM-09	When basket mode is disabled, if the selected asset balance of the receiver increased, then the honey balance of the sender strictly decreases after a successful redeem	✅	✅	1.11B+

Invariant	Description	Tested	Passed	# Runs
REDEEM-10	When basket mode is enabled, if the asset balance of the receiver for at least one registered asset has increased, then the honey balance of the sender strictly decreases after a successful redeem	✅	✅	1.11B+
REDEEM-11	When basket mode is disabled, if the redeem rates for the asset is less than 100%, the inputted amount is not 0 and the previewed redeem is not 0, then the difference in the sender's asset balance must be strictly less than the previewed deposit amount and/or the collectedFees is strictly increasing	✅	✅	1.11B+
LIQ-01	Liquidation does not unexpectedly revert	✅	❌	-
LIQ-02	If bad collateral shares of honey factory is not 0 after a successful liquidation and the inputted goodAmount is not 0, then the caller's good collateral balance decreases by exactly the inputted goodAmount	✅	✅	1.10B+
LIQ-04	The caller's bad collateral balance increases by an amount less than or equal to the inputted goodAmount multiplied by the price of the good collateral divided by the price of the bad collateral multiplied by (1 + liquidation rate) after a successful liquidation	✅	✅	1.10B+
LIQ-05	The total assets of the good collateral vault converted to shares increases by exactly the difference in the caller's good collateral balance converted to shares after a successful liquidation (with tolerance)	✅	✅	1.10B+
LIQ-06	The total assets of the bad collateral vault decreases by exactly the difference in the caller's bad collateral balance after a successful liquidation (with tolerance)	✅	✅	1.10B+
LIQ-07	The total supply of honey does not change after a successful liquidation	✅	✅	1.10B+
LIQ-08	If the amount of shares of the selected bad asset is greater than the accrued fees, and the goodAmount is not 0, then the caller's bad asset balance strictly increases after a successful liquidation	✅	✅	1.10B+

Invariant	Description	Tested	Passed	# Runs
GLOBAL-01	The _checkInvariants function never reverts	✅	✅	1.11B+
GLOBAL-02	The _isCappedRelative function never reverts	✅	❌	-
GLOBAL-03	The _isCappedRelative function always returns true for all registered assets	✅	❌	-
GLOBAL-04	The _isCappedGlobal function never reverts	✅	❌	-
GLOBAL-05	The _isCappedGlobal function always returns true	✅	❌	-
GLOBAL-06	The _getWeights function never reverts	✅	❌	-
GLOBAL-07	The sum of all weights is less than or equal to 1e18	✅	✅	1.11B+
RECAP-01	Recapitalizing does not unexpectedly revert	✅	❌	-
RECAP-03	If the inputted amount is not 0, the total assets of the asset vault strictly increases after a successful recapitalization	✅	✅	1.11B+
RECAP-04	The total supply of honey does not change after a successful recapitalization	✅	✅	1.11B+
FEE-01	withdrawAllFees does not unexpectedly revert	✅	❌	-
FEE-02	withdrawFee does not unexpectedly revert	✅	❌	-
FEE-03	If the asset balance of the fee receiver increased, then the collected fees for the fee receiver with the selected asset is 0 after a withdrawFee call	✅	✅	1.10B+
FEE-05	If the collected fees for the fee receiver is not 0 after converting to assets for the selected asset before, then the asset balance for the fee receiver strictly increases after a successful withdrawFee call	✅	✅	1.10B+
FEE-07	If the asset balance of the fee receiver increased for an asset, then the collected fees for the fee receiver for that asset is 0 after a withdrawAllFees call for the fee receiver	✅	✅	1.10B+

Invariant	Description	Tested	Passed	# Runs
FEE-09	For each asset, if the collected fees for the fee receiver is not 0 after converting to assets before, then the asset balance for the fee receiver strictly increases after a successful withdrawAllFees call	✅	✅	1.10B+
ADMIN-01	setCollateralAssetStatus does not unexpectedly revert	✅	✅	1.11B+
ADMIN-02	setMintRate does not unexpectedly revert	✅	✅	1.11B+
ADMIN-03	setDepegOffsets does not unexpectedly revert	✅	✅	1.11B+
ADMIN-04	setForcedBasketMode does not unexpectedly revert	✅	✅	1.11B+
ADMIN-05	setGlobalCap does not unexpectedly revert	✅	✅	1.11B+
ADMIN-07	setLiquidationRate does not unexpectedly revert	✅	✅	1.11B+
ADMIN-08	setPOLFeeCollectorFeeRate does not unexpectedly revert	✅	✅	1.11B+
ADMIN-09	setRedeemRate does not unexpectedly revert	✅	✅	1.11B+
ADMIN-10	setReferenceCollateral does not unexpectedly revert	✅	✅	1.11B+
ADMIN-11	setRecapitalizeBalanceThreshold does not unexpectedly revert	✅	✅	1.11B+

Optimization

Echidna's optimization mode was run for over **49,000,000+** iterations to analyze the impact of rounding errors on the difference between the amount of honey passed to the `redeem` function and the resulting change in balance while operating in basket mode.

Invariant	# Runs	Optimized Value
The maximum difference between the amount of honey passed to the `redeem` function and the resulting change in balance.	49M+	225,679,454,155
The maximum difference, expressed as a percentage of the amount passed to the `redeem` function, calculated with a precision of up to two decimal places.	49M+	0.00%
The maximum difference in the total sum of weights returned by the `getWeights` function, compared to its theoretical value of `1e18`, observed in a simulation using four collateral assets.	49M+	3

Tolerances

Rounding errors that cause multiple invariants to break were identified during our investigation. These issues exist in the current codebase but remain within acceptable limits.

To address this, tolerances were defined for each affected invariant to ensure they fall within acceptable thresholds. The table below outlines these invariants along with their respective tolerances.

Invariant	Description	Tolerance
`LIQ-05`	The total assets of the good collateral vault converted to shares increases by exactly the difference in the caller's good collateral balance converted to shares after a successful liquidation (with tolerance)	1 collateral token converted to 18 decimals
`LIQ-06`	The total assets of the bad collateral vault decreases by exactly the difference in the caller's bad collateral balance after a successful liquidation (with tolerance)	1

Disclaimer

All activities conducted by Perimeter in connection with this project were carried out in accordance with the terms outlined in a Statement of Work and an agreed-upon project plan, as set forth in a proposal document delivered prior to the commencement of the project.

Security assessment projects are subject to time limitations, and as such, the findings presented in this report should not be interpreted as an exhaustive or comprehensive identification of all security issues, vulnerabilities, or defects within the target codebase. Perimeter makes no representations or warranties that the target codebase is free from defects.

Furthermore, this report is not intended to be, and should not be construed as, investment advice or a recommendation to participate in any financial transactions. The content herein does not constitute endorsements or recommendations for any financial decisions, securities, or investment strategies.