certora

Security Assessment & Formal Verification Final Report

Bamm

October 2024

Prepared for Frax

certora

Table of content

Project Summary

Project Scope

Project Name	Repository (link)	Latest Commit Hash	Platform
Bamm	https://github.com/FraxFinance/dev-frax-bamm/	f2290e32af6d321f3bd935d1b8a3b5ed63150775	EVM

Project Overview

This document describes the specification and verification of Frax Bamm using the Certora Prover and manual code review findings. The work was undertaken from **2024-09-30** to **2024-10-21**

The following contract list is included in our scope:

```
contracts/BAMM.sol
contracts/BAMMERC20.sol
contracts/BAMMHelper.sol
contracts/BAMMUIHelper.sol
contracts/FraxswapOracle.sol
contracts/VariableInterestRate.sol
contracts/Factory/BAMMFactory.sol
```

The Certora Prover demonstrated that the implementation of the **Solidity** contracts above is correct concerning the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts. During the verification process and the manual audit, the Certora team discovered bugs in the Solidity contracts code, as listed on the following page.

Please note that a few more formal rules are not included in this report, as they were proven with an unreleased version of the Certora Prover. Once those rules are proven on a released version of the Certora Prover, we will add them to the next version of this document.

Protocol Overview

The Bamm protocol is a wrapper for a UniswapV2-like swap pair AMM, called FraxSwapPair. It allows users to borrow tokens over the bamm. The Bamm wraps the liquidity tokens and allows users to directly mint the liquidity tokens into the Bamm to reap rewards accrued from borrowing actions taken.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	-	-	-
High	-	-	-
Medium	3	-	-
Low	8	-	-
Informational	5	-	-
Total	16	-	-

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
			Likelihood	

certora

Detailed Findings

ID	Title	Severity	Status
M-01	Liquidation is capped to rent minus fee, preventing full liquidation	Medium	Fixed
M-02	Slippage check doesn't account for fee, throttling liquidation under some cases	Medium	Fixed
M-03	Borrowing from gifted pair tokens can destabilize a bamm	Medium	Fixed
L-01	Slippage check might increase the repayment percentage	Low	Fixed
L-02	Liquidation with high liquidation percentage might leave a small position that's not worth liquidating in the future	Low	Acknowledged
L-03	BAMM check for protocol fess in the pair isn't accurate	Low	Fixed
L-04	microLiquidations are not possible for pairTokens younger than 30 minutes	Low	Acknowledged
L-05	wrong rounding of lpTokenAmount in vaultSync	Low	Fixed

L-06	mintFee in addInterest may mint more than the interest	Low	Fixed
L-07	the interest fee is computed with the wrong multiplier	Low	Fixed
L-08	wrong rounding of sqrtBalance/sqrtRentedReal in addInterest	Low	Fixed

Medium Severity Issues

M-01 Liquidation is capped to rent minus fee, preventing full liquidation

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: BAMM.sol	Status: Not Fixed	

Description: At the liquidation function the rent to be repaid is capped to the rent in the vault. But this capping is done before the fee is subtracted from the repay amount. Meaning the repay amount is effectively capped to the vault's rent minus the fee, preventing full liquidation.
The remaining amount would either be marked as bad debt (if the `ltv` after liquidation is above 1) or remain as a small position that's not worth to liquidate.

JavaScript
```
        if (rentToLiquidity > vault.rented) rentToLiquidity = vault.rented;

        if (rentToLiquidity > 0) {
            int256 liquidationFee = int256(
                _ltv > SOLVENCY_THRESHOLD_FULL_LIQUIDATION
                    ? LIQUIDATION_FEE
                    : (LIQUIDATION_FEE * (_ltv - SOLVENCY_THRESHOLD_LIQUIDATION)) /
                        (SOLVENCY_THRESHOLD_FULL_LIQUIDATION -
SOLVENCY_THRESHOLD_LIQUIDATION)
            );
            rentToLiquidity = (rentToLiquidity * (1_000_000 - liquidationFee)) / 1_000_000;
```

Exploit Scenario:

- Bob takes a loan from the BAMM
- Interest accrues over time and the `ltv` reaches 99.1%

- Alice liquidates Bob's vault
    - Only 99% of the rent is liquidated due to this bug
    - The remaining 1% has an ltv that's greater than 1, therefore it's marked as bad debt and the remaining balance is added to the fee
    - Alice got ~2% of the liquidation as a fee (instead of 1%), at the expense of BAMM shareholders

Recommendations: Cap the repay to the rent after subtracting the fee

Customer's response: Acknowledged, and reported before.

Fix Review: Properly fixed Here: #768 Bamm.sol.

M-02 Slippage check doesn't account for fee, throttling liquidation under some cases

Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: BAMM.sol	Status: Not Fixed	

Description: The BAMM's liquidation function checks for slippage, and in case the slippage is above 1% it limits the repayment percentage to repay only from the swapped amount.
However, this check doesn't account for fees, so in case that the fee is set to 1% this check would be triggered for every liquidation, effectively throttling the liquidation - requiring multiple iterations and reducing the liquidation fee per iteration (even when the sell percentage goes up to 100% we'll still have something remaining to swap after every iteration, since `getMaxSell()` doesn't account for the fees as well)

```C/C++
if (token1Out < (sellToken0 * reserve1 * 99) / (uint256(reserve0) * 100)) {
    // Only repay from swap amount when there is a lot of slippage
    _repayPercentage = (token1Out * PRECISION) / (token1Out + uint256(vault.token1));
}
```

Exploit Scenario:

- BAMM is deployed for a FraxSwap pair that has 1% fees
- Positions on the BAMM become insolvent
- The fees per liquidation are too low, so nobody liquidates the positions
- The BAMM runs into bad debt since nobody liquidates it

Recommendations: Account for fees when checking for slippage

Customer's response: Acknowledged.

Fix Review: Properly fixed Here, assuming that the maximum fees would stay capped at 1%.

M-03 Borrowing from gifted pair tokens can destabilize a bamm

Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: BAMM.sol	Status: Not Fixed	

Description: Under the current implementation, It is possible to gift pair lp tokens to the bamm and borrow against that liquidity. This is specifically relevant before any initial minimal liquidity is set.

This decoupling between the pairToken balance and the bammErc20 token can potentially cause ratio attacks on the value of the bammErc20 shares.

Exploit Scenario: A fresh bamm is created through the factory.

Before initial minting, an attacker gifts some Lp tokens to the bamm and Borrow against them, creating a discrepancy between the totalSupply and the sqrtRented value. This discrepancy can lead to some unfairness or perhaps DOS vector attacks on the bamm through inflated bammERC20 price.

Note: Full attack vectors and implications have not been fully mapped out therefore there might be some additional implications yet discovered.

Recommendations:

1. Adding a check to _borrow That there is some minimum liquidity minted.

2. Instead of working on the balance of the pair token, potentially working off a storage variable that is changed through a dedicated API would allow for a finer coupling between the liquidity that the contract holds and the bammErc20 token.

Customer's response: Acknowledged that this state is unintended, but it's probably not problematic. Would fix it.

Fix Review: Properly fixed Here, by not allowing a borrow action before first minting. This fix seems to be sufficient as this is an attack vector with no exploit path. It should still be noted that scenarios relating to gifting LP tokens directly might need to be accounted for in the future.

Low Severity Issues

L-01 Slippage check might increase the repayment percentage		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: BAMM.sol	Status: Not Fixed	

Description: During liquidation the function checks for slippage - if there's any slippage it sets the repay percentage to be the amount that we get out of the swap.
This is supposed to limit the repayment percentage so that the liquidation doesn't worsen the ltv.
However, in some cases this might increase the repayment percentage, increasing the fee paid for the liquidation at the expense of the liquidated user.

```JavaScript
                if (token1Out < (sellToken0 * reserve1 * 99) / (uint256(reserve0) * 100)) {
                    // Only repay from swap amount when there is a lot of slippage
                    _repayPercentage = (token1Out * PRECISION) / (token1Out +
uint256(vault.token1));
                }
```

Exploit Scenario:

- Bob takes a loan from the BAMM
  - The reserves on the AMM are 1e25 for each token
  - Bob takes a loan of:
    - 3e23 rent

        - 97e22 collateral for token0
        - 9.7e22 collateral for token1
- Bob's vault accrues interest over time and reaches an ltv of 0.9851
- Alice liquidates Bob's position
    - The repay percentage needs to be 22%
    - Due to the slippage check it's set to 65%
    - Rent after is 1.1e23, meaning ~64% of the rent was paid
- Bob ended up paying about 3 times the fee that they would've paid without the slippage

Recommendations: Never increase the repay percentage when the slippage check passes

Customer's response: Acknowledged.

Fix Review: Properly Fixed Here.

certora

L-02 Liquidation with a high liquidation percentage might leave a small position that's not worth liquidating in the future

Severity: **Low**	Impact: **Med**	Likelihood: **Low**
Files: BAMM.sol	Status: Not Fixed	

Description: In case of liquidation with a high repay percentage (e.g. 95%) this might leave a small position that's not worth liquidating in the future. This position might accrue bad debt in the future, making the accounting inconsistent.

Exploit Scenario:

- Bob has a position that has reached 98.97% which warrants a 95% liquidation
- Alice liquidates the position
- 5% of the original position remains, the max fee of this would be 1/20 of the original position
- The position accrues interest and bad debt, but nobody liquidates it since the fee isn't worth it

Recommendations: In case the repayment percentage crosses some threshold, round it up to a full liquidation
This threshold should take into account the fee under current liquidation, the max fee under full liquidation of the remaining percentage, and the possible gas and native token price changes on the network in which it's being deployed.

Customer's response: Acknowledged, would not be fixed as this is a rounding error with low impact.

Fix Review: Not fixed.

certora

L-03 BAMM check for protocol fees in the pair isn't accurate

Severity: **Low**	Impact: **Med**	Likelihood: **Low**
Files: bamm.sol	Status: Not Fixed	

Description: The BAMM relies on the `kLast` parameter to check if the FraxSwap pair is charging any mint fees. However, there can be a scenario where the fee was turned on and then back off, in that case, the `kLast` would turn zero only on the next mint or burn that's executed in the pair.
In the meantime, the BAMM would assume that fees are on and would underestimate the total supply of the pair.

```
JavaScript
   uint256 kLast = pair.kLast();
       if (kLast != 0) {
           kLast = Math.sqrt(kLast);
           if (k > kLast) {
               uint256 num = totalSupply * (k - kLast);
               uint256 denom = (k * 5) + kLast;
               tsAdjustment = num / denom;
           }
       }
```

Exploit Scenario:

- Fees are turned on
- `kLast` is set to the last k value
- `k` grows in the meantime
- Fees are turned off
- Bob calls `BAMM.redeem()`
- Bob gets more AMM tokens than he should since the BAMM underestimates the value of the tokens

Recommendations: Check also the Factory's `feeTo()` to see if fees are actually on or not

Customer's response: Acknowledged, was reported before.

Fix Review: Properly fixed Here.

certora

L-04 microLiquidations are not possible for pairTokens younger than 30 minutes		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: FraxswapOracle.sol bamm.sol	Status: Not Fixed	

Description: Any pairToken younger than 30 minutes has a shorter history than the default bamm period for microLiquidation.

Exploit Scenario: The current interest rate is defined to be low enough that a position won't be able to reach bad debt before the 30-minute mark. However, If that were to change, then attacks can DOS bamms greeted for pair tokens younger than 30 minutes by creating bad debt that can't be liquidated before it's not profitable to do so.

Recommendations: Don't allow for bamms to be created for Amms that have a history smaller than the period.

Customer's response: Acknowledge, No expected real world impact.

Fix Review: Not fixed.

certora

L-05 wrong rounding of lpTokenAmount in vaultSync		
Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: BAMM.sol	Status: Not Fixed	Violated Property: P14

Description: When repaying rent, several divisions are needed to compute the amount of tokens to repay. The division that computes lpTokenAmount is not rounding against the protocol for negative values so the protocol may lose dust amounts.

Exploit Scenario: When a single swap pair tokens is worth slightly more than a single rent amount the user calls executeActionsAndSwap with `rent=-1`. The lpTokenAmount rounds down to 0 and the debt is repaid for free at the expense of the protocol. In general, if the user repays *x* + 0.99 lpToken worth of rent, he only has to pay *x* lpTokens worth of tokens.

Recommendations: Use similar code to the other divisions to decrease lpTokenAmount by one if rent is negative and lpTokenAmount was rounded wrong.

Customer's response: Acknowledged.

Fix Review: Properly fixed Here.

certora

L-06 mintFee in addInterest may mint more than the interest.

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: BAMM.sol	Status: Not Fixed	Violated Property: P14

Description: In addInterest new tokens are minted for the fee receiver for a part of the interest. Due to rounding, this fee can be larger than the expected 10% and can even lead to the BAMM losing value.

Exploit Scenario: Run addInterest when slightly below 1e-18 interest accumulated. In that case the rentMultiplier is unchanged due to rounding down, but the mintFee uses the sqrtInterest which is greater than 0 to compute the tokens. This can lead to minting tokens without receiving interest. One example is:

```
period = 1
InterestRate = 1
rentedMultiplier_before = 1e18
sqrtRented = 16.66e18
sqrtInterest = 16
rentedMultiplier_after = 1e18 (because of rounding)
fee = 1
```

Recommendations: Use the change applied to the multiplier as the basis for computing the fee. Instead of computing sqrtInterest, compute the delta of the multiplier. These are the changes to `addInterest()`:

```
uint256 interestRate = …
uint256 deltaMultiplier = rentedMultiplier_ * interestRate * period / PRECISION;
// Update the rentedMultiplier
// The original lender will get more LP back as their "earnings" when they redeem their BAMM tokens
rentedMultiplier_ = rentedMultiplier_ + deltaMultiplier;
…
        // accrue fee
        uint256 fee = deltaMultiplier * sqrtRentedAsUint * FEE_SHARE / (10000*PRECISION);
```

.

Customer's response: Acknowledged.

Fix Review: Properly fixed Here.

certora

L-07 the interest fee is computed with the wrong multiplier

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: BAMM.sol	Status: Not Fixed	Violated Property: P14

Description: In addInterest the sqrtInterest is multiplied with the new multiplier instead of the multiplier before adding interest. For very high interest the fee can even be higher than the increase in the balance.

Exploit Scenario: Only call addInterest after a long time when 500 % interest accumulated. The sqrtInterest will be 5 times the current sqrtRented. The rentedMultiplier is multiplied with 6 and the fee is 10% of 5*sqrtRented with 6*old(rentedMultiplier) making the fee 60 % of the interest that was earned by increasing the multiplier.

Recommendations: See the suggested changes for L-06; they also fix this problem.

Customer's response: Acknowledged.

Fix Review: Properly fixed Here.

certora

L-08 wrong rounding of sqrtBalance/sqrtRentedReal in addInterest

Severity: **Low**	Impact: **Low**	Likelihood: **Medium**
Files: BAMM.sol	Status: Not Fixed	Violated Property: P14

Description: When converting the interest amount into the number of tokens the interest uses rounds the current assets of the token wrongly. This can lead to creating tokens that are worth slightly more than the fee.

Exploit Scenario: This always happens when computing interest; only the fee receiver can exploit this by calling addInterest at times where the rounding is most profitable to him.

Recommendations: Copy the code from mint() that uses the right rounding methods when computing sqrtBalance/sqrtRented. This needs to be done in addition to the fix from L-06.

Customer's response: Acknowledged.

Fix Review: Properly fixed Here.

Informational Severity Issues

I-01. `getMaxSell()` doesn't account for swap fees

Description: `getMaxSell()` calculates the amount to be swapped for the ratio in the vault to match the ratio in the AMM.
However, it doesn't take into account the swap fees, which leads to an amount that's slightly lower than the desired amount.

Recommendation: Consider accounting for the fees as well

Customer's response: Acknowledge.

Fix Review: Not fixed.

I-02. Cross-contract reentrancy guard

Description: Even though the Bamm and the AMM have a reentrancy guard in place, they are not cross-protecting. If a future feature of the AMM allows for hooks, it is important to have cross-reentrency protection in place.

Recommendation: Add a view method to FraxSwapPair to read its locked status, and add a check to the reentrancy guard modifier to check that flag.

Customer's response: Acknowledge, would not redeploy FraxSwapPair for this reason alone as no issue was found, would consider if we redeploy.

Fix Review: Not fixed.

I-03. The protocol does not support ERC tokens with hooks OnReceive or onTransfer

Description: ERC tokens like ERC-777 with hooks onReceive or onTransfer are not supported by the BAMM. These tokens might be added in the future to FraxSwap, and that might cause security instabilities in the BAMM. One example might be changing the reserves mid-method call.

Recommendation: It is recommended to have clear documentation to that effect, as well as a hard comment in the code, to make sure that if this feature is added in the future it will change some of the code assumptions and make it less secure.

Customer's response: Acknowledge.

Fix Review: Not fixed.

I-04. microLiquidation might want to get the current reserves after the swap.

Description: Currently to save on gas the current microLiqudation code updates the reserves after the swap "manually", this does not take into account possible changes in the fraxSwapPair.

Recommendation: use the FraxSwapPair sync to get the correct reserves. (or the _addInterest API)

Customer's response: Acknowledge.

Fix Review: Not fixed.

I-05. microLiquidation won't work on any chain with a shorter than 12 seconds per Block

Description: The current period is 30 minutes which on mainnet is about 150 blocks. Which is less than the 1024 rounds that are checked in the Oracle. If however blocks are allowed to be passed at a more frequent frequency, then the history could be greater than the period making the oracle revert and revert the liquidation call with it.

Recommendation: add an additional round (making it 2048 blocks) to the Oracle search window.

Customer's response: Acknowledge, might only be a problem if the seconds per Block is 1 second. Was reported before and fixed.

Fix Review: Properly fixed Here.

Formal Verification

Verification Notations

Formally Verified	The rule is verified for every state of the contract(s), under the assumptions of the scope/requirements in the rule.
Formally Verified After Fix	The rule was violated due to an issue in the code and was successfully verified after fixing the issue.
Violated	A counter-example exists that violates one of the assertions of the rule.

General Assumptions and Simplifications

1. We work with objects inherited from the original contracts. In the inherited objects we add more view methods, flags, etc. In cases where it was not possible to collect the required information via the inherited object, we modified the original. These modifications don't affect the functionality of the original contracts and the verification results hold also for the original contracts.
2. We unroll loops for a fixed number of iterations. The exact number of iterations is mentioned for each contract.
3. We replaced some functions with equivalent CVL implementations. Notably *mulDiv* and *sqrt*. This speeds up the verification process.

Formal Verification Properties

FixedPoint.sol

Module General Assumptions

- We verified the contract functions against an arbitrary storage state.
- We verify the contract as a stand-alone one, i.e. we make no assumptions about the caller. We assume all methods may be called with arbitrary arguments.
- Loops are assumed to iterate at most 10 times.

Module Properties

P-01. encode - decode are inverse

Status: Verified

Rule Name	Status	Description	Link to rule report
encodeDecode_Identity	Verified	For all x: decode(encode(x)) == x	*Report*

P-02. sqrt works correctly

Status: Verified

Rule Name	Status	Description	Link to rule report
sqrt_correctBounds	Verified	For all x: decode(sqrt(x)) * decode(sqrt(x)) <= decode(x) For all x: [decode(sqrt(x))+1] * [decode(sqrt(x))+1] > decode(x)	*Report*

P-03. muluq works correctly

Status: Verified

Rule Name	Status	Description	Link to rule report
muluq_correctBounds	Verified	For all x, y: decode(x) * decode(y) <= decode(muluq(x,y)) For all x, y: [decode(x)+1]*[decode(y)+1] > decode(muluq(x,y))	*Report*

P-04. divuq works correctly

Status: Verified

Rule Name	Status	Description	Link to rule report
divuq_correctBounds	Verified	For all x, y: decode(x) / [decode(y)+1] <= decode(divuq(x,y)) For all x, y: [decode(x)+1] / decode(y) >= decode(divuq(x,y)) As long as decode(y) > 0	*Report*

certora

P-05. muluq - sqrt correspondence

Status: Verified

Rule Name	Status	Description	Link to rule report
sqrt_mul_correspondence	Verified	For all x: decode(muluq(sqrt(x),sqrt(x))) <= decode(x)	*Report*

BitMath.sol

Module General Assumptions

- We verified the contract functions against an arbitrary storage state.
- We verify the contract as a stand-alone one, i.e. we make no assumptions about the caller. We assume all methods may be called with arbitrary arguments.

Module Properties

P-06. leastSignificantBit works correctly

Status: Verified

Rule Name	Status	Description	Link to rule report
lsb_neverReverts	Verified	*leastSignificantBit* never reverts	*Report*
lsb_Correctness	Verified	For all x > 0: x & 2^lsb(x) != 0 For all x > 0: x & [2^lsb(x)-1] == 0	*Report*

P-07. mostSignificantBit works correctly

Status: Verified

Rule Name	Status	Description	Link to rule report
msb_neverReverts	Verified	*mostSignificantBit* never reverts	*Report*

certora

msb_Correctness	Verified	For all x > 0: x >= 2^msb(x) For all x > 0: x < 2^[msb(x)-1]	*Report*

BAMM.sol

Module General Assumptions

- We verified the contract functions against an arbitrary storage state.
- We use basic standard implementations for underlying ERC20 tokens.
- We verify the contract in presence of other contracts, namely `FraxswapDummyRouter, FraxswapOracle, BAMMERC20, VariableInterestRate`. This means that interactions between these contracts are modeled exactly, and we make no assumptions about interactions with other contracts.
- Loops are assumed to iterate at most 2 times.
- We assume that the contract starts in a *valid* state. Specifically:
  - *rentedMultiplier >= 1E18*
  - all vaults are *valid*
  - *sqrtRented >= 0*
  - *sqrtRented* >= the sum of *vault.rented* over any subset of *vaults*
  - *token0.balanceOf(BAMM)* >= the sum of *vault.token0* over any subset of *vaults*
  - *token1.balanceOf(BAMM)* >= the sum of *vault.token1* over any subset of *vaults*
  - *total supply* equals sum of balances for all *ERC20* tokens
- We added auxiliary methods that perform specific tasks of *ExecuteActionsAndSwap.* Namely:
  - *addToken0(uint256, address)*
  - *addToken1(uint256, address)*
  - *removeToken0(uint256, address)*
  - *removeToken1(uint256, address)*
  - *borrow(uint256, address)*
  - *repay(uint256, address)*
  - *swapToken0(uint256, address)*
  - *swapToken1(uint256, address)*

  We use these methods instead of the original *ExecuteActionsAndSwap* in properties P13 and P14.

Module Properties

P-08. mint works correctly

Status: Verified	Assumptions: interest has not been added, i.e., *timeSinceLastInterestPayment == block.timestamp*

Rule Name	Status	Description	Link to rule report
mint_integrity	Verified	*mint* correctly updates balances of *msg.sender* and user *"to"*.	Report
mint_doesntAffectOthers	Verified	*mint* doesn't change balances of users other than *msg.sender* and *"to"*.	Report

P-09. redeem works correctly

Status: Verified	Assumptions: interest has not been added, i.e., *timeSinceLastInterestPayment == block.timestamp*

Rule Name	Status	Description	Link to rule report
redeem_integrity	Verified	*redeem* correctly updates balances of *msg.sender* and user *"to"*.	Report
redeem_doesntAffectOthers	Verified	*redeem* doesn't change balances of users other than *msg.sender* and *"to"*.	Report

P-10. mint + redeem not profitable

Status: Verified

Rule Name	Status	Description	Link to rule report
mint_redeem_notProfitable	Verified	*For all x: redeem(mint(x))* <= x	*Report*

P-11. Vault is always valid

Status: Verified

Rule Name	Status	Description	Link to rule report
vaultRemainsValid	Verified	If Vault of user “u” is valid, then it is valid after any method call. Validity is determined by function *_isValidVault*.	*Report*

P-12. Correspondences between BAMM and vaults

Status: Verified

Rule Name	Status	Description	Link to rule report
sqrtRented_neverLTzero	Verified	If *BAMM.sqrtRented >= 0* then it will stay non-negative after any method call.	*Report*
sqrtRentedEqualsSumVaultRented	Verified	*BAMM.sqrtRented ==* sum of *vault.rented* over all vaults.	*Report*

P-13. rentedMultiplier doesn't decrease in time

Status: Verified

Rule Name	Status	Description	Link to rule report
rentedMultiplier_neverDecreases	Verified	RentedMultiplier cannot be decreased by any of the following methods: *redeem(address,uint256), mint(address,uint256), addInterest(), addToken0(uint256,address), swapToken1(uint256,address), removeToken0(uint256,address), repay(uint256,address), swapToken0(uint256,address), addToken1(uint256,address), removeToken1(uint256,address), borrow(uint256,address), microLiquidate(address)*	*microLiquidate* *other methods*

P-14. BAMM share value doesn't decrease

Status: Violated

Rule Name	Status	Description	Link to rule report
bammSharesValueAlwaysIncreases	Violated	BAMM share value cannot be decreased by any of the following methods: *redeem(address,uint256), mint(address,uint256), addInterest(), addToken0(uint256,address), swapToken1(uint256,address), removeToken0(uint256,address), repay(uint256,address), swapToken0(uint256,address), addToken1(uint256,address), removeToken1(uint256,address), borrow(uint256,address),* For each of these methods we prove that the value of *(sqrtBalance + sqrtRentedReal) / bamm.totalSupply()* after the method is at least as high as the value before the method call.	*Repay* *Borrow* *AddInterest* *AddInterest*

Disclaimer

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.