Nome

# FINAL REPORT

February ‘2025

## Disclaimer:

## 1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

| Project | Nome Protocol Audit |
|---|---|
| Website | nome.gg |
| Language | Solidity |
| Methods | Manual Analysis |
| Github repository | https://github.com/nome-protocol/nome-contracts/tree/cc3550e205ce2b429f1d3c634825f400af79f224/contracts |
| Resolution 1 | https://github.com/nome-protocol/nome-contracts/tree/70c00e682150d07d7a48d5116bff66367dde7be4/contracts |

## 2. Detection Overview

| Severity | Found | Resolved | Partially Resolved | Acknowledged (no change made) | Failed resolution |
|---|---|---|---|---|---|
| **High** | 5 | 4 |  | 1 |  |
| **Medium** | 3 |  |  | 3 |  |
| **Low** | 12 |  |  | 12 |  |
| **Informational** | 9 |  |  | 9 |  |
| **Governance** | 2 |  |  | 2 |  |
| **Total** | 31 | 4 |  | 27 |  |

## 2.1 Detection Definitions

| Severity | Description |
|---|---|
| **High** | The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users. |
| **Medium** | While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences. |
| **Low** | Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately |
| **Informational** | Effects are small and do not post an immediate danger to the project or users |
| **Governance** | Governance privileges which can directly result in a loss of funds or other potential undesired behavior |

## 3. Detection

During the audited commit, only reward tokens were at risk. As long as the staking token was different to the reward token, no user funds were at risk.

## Global

| Issue_01 | Multiple functions lacking necessary event emissions |
|---|---|
| **Severity** | **Informational** |
| **Description** | There are numerous instances throughout the project where event emissions containing useful or necessary information are missing completely, e.g:<br>- StabFund's deposit and withdraw<br>- Oracle's adding and activating operators, pausing/unpausing<br>- Boardroom's setGauge |
| **Recommendations** | Consider implementing correct events when changing critical state |
| **Comments / Resolution** | Acknowledged. |

| Issue_02 | Outdated Solidity version |
|---|---|
| **Severity** | **Informational** |
| **Description** | The project currently runs on Solidity's 0.6.6 version, which is fairly outdated and could introduce complications due to compiler bugs, which may impact new functionality added to the system later on. |
| **Recommendations** | Consider using the latest compiler version as they are most optimized and safe. It has to be noted that with 0.8.0 overflow is explicitly prevented. In most codebases this is not a problem. However, for example within UniswapV3's feeGrowth calculation, this has already introduced multiple bugs. |
| **Comments / Resolution** | Acknowledged. |

## EmissionManager

The EmissionManager contract is responsible for executing positive rebases. A rebase is possible if the price of the synthetic token has surpassed the threshold. During these rebases, the contract will split up the minting of additional synthetic tokens to 6 addresses based on the set percentage for each address. These addresses include the stableFund, devFund, gauge, lockBoardroom, liquidBoardroom, and bexBoardroom.

The contract inherits IEmissionManager, ReentrancyGuard, Operatable, Debouncable, and Timeboundable.

**Debouncable:**
The Debouncable contract allows the owner to set the debouncePeriod, this period is the amount of time anyone must wait before calling the same function again. This functionality is written in the modifier debounce. The debounce modifier has been added to the makePositiveRebase function in order to add a sort of cool down period before being able to call the function again.

**Timeboundable:**
The Timeboundable contract allows the owner to specify a _start and _finish timestamp. These timestamps are used in the inTimeBounds modifier which assures that the current timestamp is greater than or equal to the start timestamp and that the current timestamp is less than or equal to the finish timestamp. Currently the makePositiveRebase makes use of the inTimeBounds modifier to ensure calling of this function occurs during the desired timestamps.

**Core Invariants:**

INV 1: An address should not be minted tokens if their rate is not greater than 0 or the address is address(0)

INV 2: Rebases cannot occur more frequently than the debounce period.

INV 3: Rebases cannot occur if the threshold for the rebase has not been reached.

INV 4: The rebase amount cannot be greater than maxRebase.

INV 5: devFundRate and stableFundRate must be non zero in order to trigger a rebase.

INV 6: tokenManager, stableFund, devFund, bexBoardroom, and liquidBoardroom must have set addresses in order to trigger a rebase.

**Privileged Functions**

- setDevFund
- setGauge
- setStableFund
- setLiquidBooardroom
- setLockBoardroom
- setBexBoardroom
- setTokenManager
- setDevFundRate
- setGaugeRate
- setGaugeIncetiveRate
- setStableFundRate
- setLockBoardroomRate
- setLiquidBoardroomRate
- setBexBoardroomRate
- setThreshold
- setStabFundBalanceThresholdUnits
- setMaxRebase
- setRebasePeriod
- setPausePositiveRebase
- transferOperator

| Issue_03 | EmissionManager::makePositiveRebase may be dosed using flashloans |
|---|---|
| **Severity** | **High** |
| **Description** | The makePositiveRebase function makes a call to positiveRebaseAmount in order to return the rebase amount for each token. However a check in the function may allow a user to dos the function:<br><br>`uint256 currentPriceUndPerUnitSyn =`<br>`    tokenManager.currentPrice(syntheticTokenAddress, oneSyntheticUnit);`<br><br>`if (rebasePriceUndPerUnitSyn < thresholdUndPerUnitSyn) {`<br>`    return 0;`<br>`}`<br>`// Extra safe check!`<br>`if (currentPriceUndPerUnitSyn < thresholdUndPerUnitSyn) {`<br>`    return 0;`<br>`}`<br><br>In specific, let's note that we call tokenManager.currentPrice.<br>This function will query the balancer pool.<br><br>Using flashloans a malicious user can first use the loan to temporarily change the price of the pool in order to not pass the check above, then the user will call makePositiveRebase, since the check does not pass then we return 0.<br>Given that the function has the debounce modifier, it will be dosed by the debounce period. Which is 24 hours from looking at the test files. Therefore a malicious user can potentially DOS the makePositiveRebase function indefinitely. |
| **Recommendations** | Consider only relying on the TWAP for checks. |
| **Comments / Resolution** | Resolved by following recommendations. |

| Issue_04 | makePositiveRebase will revert in the notifyTransfer when there are no stakes, which will prevent a rebase |
|---|---|
| **Severity** | **Low** |
| **Description** | In the _makeOnePositiveRebase the notifyTransfer function is called for every boardroom. However this may cause the rebase to fail due to the following check in the notifyTransfer<br><br>*require(*<br>*shareSupply > 0,*<br>..<br><br>As a result the notifyTransfer will temporarily prevent the _makeOnePositiveRebase from execution |
| **Recommendations** | Consider implementing an initial stake from the owner for each boardRoom so that there are always non-zero shares. |
| **Comments / Resolution** | Acknowledged. |

| Issue_05 | addIncentive may prevent a rebase in an edge case |
|---|---|
| **Severity** | **Low** |
| **Description** | In the _makeOnePositiveRebase the addIncentive function is called to add incentive for the gauge. However addIncentive may revert for some tokens if the amount passed is less than the minIncentiveRate specified in the gauge:<br><br>*if (amount < minIncentiveRate)*<br>*AmountLessThanMinIncentiveRate.selector.revertWith();*<br><br>As a result the addIncentive may prevent the rebase if one of the managed tokens has a higher minIncentiveRate and a lower totalSupply. This will temporarily prevent all tokens from rebasing until the amount reaches the minIncentiveRate or the owner |

| | |
|---|---|
| | changes the gaugeRate. |
| **Recommendations** | Consider to ensure that the amount passed in the addIncentive function is more than the minIncentiveRate. |
| **Comments / Resolution** | Acknowledged. |

| Issue_06 | maxRebase is impractical in a case of extreme depeg |
|---|---|
| **Severity** | **Low** |
| **Description** | The positiveRebaseAmount function executes rebases based on a threshold and a maximum value for rebasing in order to keep price movements controlled. The value is set in place to handle more extreme cases in a controlled manner, but it is impractically hardcoded and could leave the price still depegged.<br>In the scenario where the maximum allowed value would still leave the price over the threshold, another rebase cannot be initiated for the next 6 hours. |
| **Recommendations** | Consider computing a value that satisfies the threshold to set for the maximum, hardcoding it could lead to an unfavorable state. |
| **Comments / Resolution** | Acknowledged. |

| Issue_07 | Rate updating functions do not validate the total distribution rate is not above 100% |
|---|---|
| **Severity** | **Low** |
| **Description** | The owner function for updating the percentile distribution of rebased tokens:<br>- setDevFundRate<br>- setGaugeRate<br>- setStableFundRate<br>- setLockBoardroomRate<br>- setLiquidBoardroomRate<br>- setBexBoardroomRate<br>do not sanitize that the total value from all of them amounts to 100%.<br>Accidentally going over the 100 percentile would lead to minting more synthetic tokens than necessary. |
| **Recommendations** | Consider introducing a computation, preferably a modifier, that calculates the new sum of the rates and keeps it equal to 100%. |
| **Comments / Resolution** | Acknowledged. |

| Issue_08 | makePositiveRebase will fail if a new token is inserted into TokenManager's tokens array |
|---|---|
| **Severity** | **Low** |
| **Description** | In EmissionManager, when makePositiveRebase is called, it calls tokenManager.allTokens and loops through each token in the tokens array. The function then goes on to call tokenManager.averagePrice, which calls the oracles getTWAP.<br><br>In Oracle.getTWAP, the token must have at least 2 valid snapshots, otherwise the function will revert.<br><br>If a new token is added in the TokenManager through addToken, makePositiveRebase will not work until there are enough valid |

| | |
|---|---|
| | snapshots from the new token's oracle.<br><br>This will delay the rebasing process. |
| **Recommendations** | Consider checking that the token's oracle has enough snapshots (getTWAP works successfully) before being able to add the token into the tokens array in the Token Manager contract. |
| **Comments / Resolution** | Acknowledged. |

| Issue_09 | makePositiveRebase can reach out of gas errors |
|---|---|
| **Severity** | **Informational** |
| **Description** | The function takes all the tokens in the tokens array and calls _makeOnePositiveRebase in a loop. The function also calls the getTWAP function which loops over all MAX_SNAPSHOTS and transfers tokens to different places.<br><br>Depending on the window size, there can be many iterations and if there are many tokens to loop through, the gas cost may be too high which can lead to out of gas errors. |
| **Recommendations** | Consider the risk associated with a large amount of tokens in the tokenManager. With too many tokens, the function makePositiveRebase will be dosed. |
| **Comments / Resolution** | Acknowledged. |

| Issue_10 | Off by one error in EmissionManager in stabFundBalanceThresholdUnits |
|---|---|
| Severity | **Informational** |
| Description | The first if statement of the positiveRebaseAmount function in EmissionMananger checks that the balance of the stable fund is not greater than or EQUAL to the stabFundBalanceThresholdUnits LN122.<br>However the code comments from stabFundBalanceThresholdUnits states this:<br><br>*If stab fund has more than 'stabFundTokensThreshold' tokens, then rebase is not possible. LN 44*<br><br>This shows that the check should just be greater than instead of greater than or equal to. |
| Recommendations | Consider fixing the off-by-one error, alternatively this issue can be acknowledged. |
| Comments / Resolution | Acknowledged. |

| Issue_11 | Comments in EmissionManager are different from code |
|---|---|
| Severity | **Informational** |
| Description | In Line 147, maxRebase is commented as 2e18, but the initialized maxRebase value is 300, which makes it 3e18 instead |
| Recommendations | Consider changing the comment or removing it. |
| Comments / Resolution | Acknowledged. |

## TokenManager

The TokenManager contract serves as a hub for synthetic token integration in the protocol. The TokenManager allows the operator to add or remove tokens which will be used for rebases in the EmissionManager. Additionally the contract serves as an entry point to the oracle for the EmissionManager. This contract also exposes key functions that are called in the EmissionManager, these functions include oneSyntheticUnit, oneUnderlyingUnit, averagePrice, currentPrice, allTokens, and mintSynthetic.

The contract can get both the current and average price for a synthetic token address. The average price is returned by making a call to the oracle function getTWAP. The currentPrice is returned by making a call to querySwap to the balancer pool.

**Core Invariants:**

INV 1: addToken,deleteToken, burnSyntheticFrom, and mintSynthetic cannot be called until an emissionManager is set.

INV 2: Setting a new emissionManager will remove token admin rights from the previous emissionManager

INV 3: A new operator is set upon deletion of a synthetic token from the token manager.

**Privileged Functions**

- addTokenAdmin
- deleteTokenAdmin
- addToken
- deleteToken
- burnSyntheticFrom
- mintSynthetic
- setEmissionManager
- setOracle
- migrateBalances
- transferOperator

| Issue_12 | Arrays are incorrectly handled, as deleting entries does not reduce array size, leading to unbound looping |
|---|---|
| **Severity** | **Low** |
| **Description** | Both _deleteTokenAdmin and deleteToken execute a loop over the entire data arrays in order to find the key to be removed.<br><br>However those functions simply invoke delete on the variable, which simply resets its value to 0 and does not pop it, keeping the array size the same.<br><br>In both cases, when adding new admins and tokens, the arrays grow with no ability to reduce their size, opening up the potential for OOG errors during looping. |
| **Recommendations** | Consider using a swap and pop mechanism to safely remove array entries, by swapping the selected index with the last one and popping, effectively removing the element to delete. |
| **Comments / Resolution** | Acknowledged. |

## BexBoardroom

The BexBoardroom contract extends the boardRoom contract functionality by allocating reward emissions to lp Pool stakers. This is done by adding the lp Pool balance of the user to shareTokenBalance. Additionally the lp Pool total supply is added to the stakingTokenSupply in the shareTokenSupply function in order to ensure proper tracking of rewards.

This extended functionality will distribute token emission among shareholders that stake NOME and lock NOME in lpPool. This incentivizes users to stake tokens on the lp Pool.

**Core Invariants:**

INV 1: lpPool must be set first in order to interact with the contract

**Privileged Functions**

- transferOwnership
- renounceOwnership
- transferOperator
- notifyTransfer
- setTokenManager
- setEmissionManager
- setPause
- setLpPool

| Issue_13 | WITHDRAW_DELAY_BLOCKS will be bypassed allowing theft of rewards |
|---|---|
| **Severity** | **High** |
| **Description** | BexBoardroom will allow users who stake in the RewardsPool to be eligible for USDbr rewards.<br><br>In the Boardroom contract there is protection against flash loans to prevent users from extracting more rewards without having participated in the stake:<br><br>*require(*<br>    *block.number >= lastEmissionBlock.add(WITHDRAW_DELAY_BLOCKS),*<br>    *"Boardroom: must wait 100 blocks after last emission to withdraw"*<br>  *);*<br><br>However if users use the RewardsPool.sol to stake and withdraw, this WITHDRAW_DELAY_BLOCKS is not implemented.<br><br>By omitting this, an attacker will be able to use a flash loan, deposit in a pool to receive Lp tokens, stake them in the RewardsPool, and call the makePositiveRebase function. After that they will withdraw and return the loan. This will allow him to receive the majority of the rewards without having participated in the stake |
| **Recommendations** | Consider enforcing flashloan protection in the RewardsPool as well. |
| **Comments / Resolution** | Resolved by following recommendation. |

| Issue_14 | BexBoardroom will accrue unclaimable rewards if users stake on RewardsPool before boardroom is set |
|---|---|
| **Severity** | **Medium** |
| **Description** | The bexBoardroom contract adds the lpPool total supply to the shareTokenSupply in order to allow users who stake LP(in RewardsPool) to benefit from reward emissions.<br><br>`function shareTokenSupply() public view override returns (uint256) {`<br>`    return stakingTokenSupply.add(lpPool.totalSupply());`<br><br>However this leads to stuck rewards in the contract that cannot be claimed by anyone. This happens because we use the shareTokenSupply in notifyTransfer to calculate the deltaRPSU<br><br>`    uint256 shareSupply = shareTokenSupply();`<br>`...`<br>`    uint256 deltaRPSU = amount.mul(stakingUnit).div(shareSupply);`<br><br>Because of this, users who have a balance of staked lp tokens but have not yet interacted with bexBoardroom( i.e. staked before the boardroom is set on RewardsPool) will not be able to claim the rewards allocated to them for being staked lp token holders. This happens because they will not have any previous snapshots and thus will not be able to claim the rewards. Furthermore this will lower the reward amount for others and the remaining tokens will remain unclaimable.<br><br>The root of this issue stems from the rewardPool not setting the boardroom address in the constructor. Thus users will be given lp tokens without calling updateAccruals if staking before the boardroom address is set.<br><br>`function _updateBoardroomAccruals(address owner) internal {`<br>`    if (address(boardroom) != address(0)) {`<br>`        boardroom.updateAccruals(owner);` |

|  |  |
|---|---|
|  | } |
| **Recommendations** | Consider preventing staking on RewardsPool if the boardroom has not yet been set.<br><br>Alternatively, the deployment process can be optimized and this issue acknowledged. |
| **Comments / Resolution** | Acknowledged, configuration scripts will ensure the boardroom is set. |

## LiquidBoardroom

The LiquidBoardRoom inherits the boardroom contract with no changes. The boardroom contract is a staking contract, allowing a user to stake a token in order to accrue rewards over time. The contract allows staking for another address and is not restricted to staking for msg.sender. The contract also allows withdrawing to a different address, however the balances are updated for the msg.sender.

**Core Invariants:**

INV 1: Every user interaction with state should update accruals

INV 2: User cannot withdraw more than staked

INV 3: Users should only be able to withdraw during the paused state.

**Privileged Functions**

- transferOwnership
- renounceOwnership
- transferOperator
- notifyTransfer
- setTokenManager
- setEmissionManager
- setPause

No issues found.

## Boardroom

The BoardRoom contract is a staking contract, allowing a user to stake a token in order to accrue rewards over time. The contract allows staking for another address and is not restricted to staking for msg.sender. The contract also allows withdrawing to a different address, however the balances are updated for the msg.sender.

**Core Invariants:**

INV 1: Every user interaction with state should update accruals

INV 2: User cannot withdraw more than staked

INV 3: Users should only be able to withdraw during the paused state, staking is not allowed during the paused state.

**Privileged Functions**

- transferOwnership
- renounceOwnership
- transferOperator
- notifyTransfer
- setTokenManager
- setEmissionManager
- setPause

| Issue_15 | Malicious user can drain rewards from Boardroom |
|---|---|
| **Severity** | **High** |
| **Description** | Boardroom::stake updates rewards accrual for msg.sender but adds the staked amount to the to address.<br><br>*function stake(address to, uint256 amount)*<br>*...*<br>*{*<br>*...*<br>*updateAccruals(msg.sender);*<br>*stakingTokenBalances[to] =*<br>*stakingTokenBalances[to].add(amount);*<br>*...*<br>*}*<br><br>This means that rewards accrual for the to address is not updated but they now have a staking balance.<br><br>A malicious user can now call the claimRewards using the to wallet. Since the address accruals are not updated (i.e the lastSnapshotId is 0, indicating that amount has been staked since snapshot 0, which is incorrect ) they will be eligible for way more rewards. |
| **Recommendations** | Consider updating accruals for the to address instead of msg.sender. |
| **Comments / Resolution** | Resolved by following recommendations. |

| Issue_16 | Users are not incentivized to stake for the whole duration |
|---|---|
| **Severity** | **Medium** |
| **Description** | Due to snapshot based staking users will be able to stake only for 101 blocks (approximately 3 minutes) and earn the same rewards as users who were staking for the whole duration of the debounce period(6 hours).<br><br>This is unfair since the makePositiveRebase function is predictable and most users will only stake, once a rebase is about to happen. Honest users who are providing long-term liquidity, will receive less rewards due to having to share their rewards with short term stakers. |
| **Recommendations** | Consider using time-based staking. This requires refactoring the contract and a re-audit. |
| **Comments / Resolution** | Acknowledged, design choice. |

## DevFund

The DevFund contract receives minted synthetic tokens by the emissionsManager during a positive rebalance. The dev fund functionality is mainly the depositing and withdrawing of tokens. The deposit function is not restricted allowing anyone to deposit tokens to the dev fund. However the withdraw function is only callable by the operator to ensure that tokens are not transferred out without proper permissions.

**Core Invariants:**

INV 1: Only the operator can withdraw tokens

INV 2: Anyone can deposit tokens

**Privileged Functions**

- transferOwnership
- renounceOwnership
- transferOperator
- withdraw

No issues found.

## StabFund

The StabFund contract receives the majority of emissions from the EmissionManager and is in charge of keeping the price pegged or as close to $1. This is done via selling of tokens when price is above $1 and buying and burning of tokens when the price is below $1. The stabFund only includes permissioned functions which ensure that only the protocol team or associates can interact to ensure stability of the price.

The StabFund contract includes 3 permissioned actors, the operator, the trader, and the owner. The operator is allowed to add or remove traders. Traders are allowed to swap tokens, deposit, and withdraw from vaults. The owner is allowed to add/remove tokens and vaults, migrate token balances, and transfer operators.

**Core Invariants:**

INV 1: Only the owner is allowed to remove/add tokens and vaults.

INV 2: Only traders are allowed to swap tokens and deposit/withdraw from vaults.

INV 3: Only the operator can add or remove traders.

INV 4: The token or vault must be allowed for swaps, deposits, and withdrawals.

**Privileged Functions**

- transferOwnership
- renounceOwnership
- swapExactTokensForTokens
- deposit
- withdraw
- approve
- addTrader
- deleteTrader
- addToken
- deleteToken
- addVault
- deleteVault
- approveToken
- transferOperator

- migrateBalances
- migrateOwnership

| Issue_17 | Any trader can steal all the funds from the stabilization module |
|---|---|
| **Severity** | **Governance** |
| **Description** | Traders are responsible for maintaining stability and control of all the assets in the StabFund contract.<br>Simply by using approve and then placing a different recipient address in the swapExactTokensForTokens any trader will be able to steal funds from this contract, therefore they should be operated with extreme caution. |
| **Recommendations** | Consider incorporating a Gnosis Multisignature contract as owner and ensuring that the Gnosis participants are trusted entities. |
| **Comments / Resolution** | Acknowledged. |

| Issue_18 | Modifiers may dos StabFund functions due to ever-increasing unbounded arrays |
|---|---|
| **Severity** | **Low** |
| **Description** | In StabFund the modifiers onlyTrader, onlyAllowedVault, onlyAllowedTokens, loops through the allowedTokens, allowedVaults and allowedTraders array.<br><br>The issue is that these arrays only increase in size indefinitely as tokens, vaults or traders are added (i.e. even when a trader/vault/token is removed the array size stays the same but adding new members will increase the array size, for example there might only be 50 allowed traders but the array size might be say 200 due to empty slots from previously deleted traders).<br>Overtime as tokens/vaults/traders are added and removed, this can result in a dos |

| | |
|---|---|
| | It's worth noting that some functions make use of all these modifiers combined eg stabFund::deposit, increasing the risk. |
| **Recommendations** | Consider implementing another solution.<br><br>An address to bool mapping will be a better solution.<br>or an enumerable mapping is we still require a list of valid tokens, vaults and traders.<br><br>Alternatively, this issue can also be acknowledged by ensuring reasonable additions. |
| **Comments / Resolution** | Acknowledged. |

| Issue_19 | swapExactTokensForTokens does not actually enforce the exactTokens to tokens path |
|---|---|
| **Severity** | **Low** |
| **Description** | The current iteration of swapExactTokensForTokens in the StabFund assumes that the trader is actually swapping using the exactTokens to tokens path. |
| **Recommendations** | The function uses IBalancerVault.swap() which takes in several parameters, one of them is singleSwap which has a value called SwapKind. The SwapKind struct has 2 values, GIVEN_IN and GIVEN_OUT.<br><br>Consider to enforce the token path, make sure GIVEN_IN is always enforced in the logic.<br><br>Alternatively, this issue can also be acknowledged. |
| **Comments / Resolution** | Acknowledged. |

## RewardsPool

The RewardsPool contract integrates with the BexBoardroom contract and inherits the stakingRewards contract. Since the BexBoardroom allows stakers of the RewardsPool to receive token emissions, the contract has extended functionality of the stakingRewards contract to correctly integrate with BexBoardroom.

The updated functionality includes updating accruals on the BexBoardroom for both stakes and withdrawals. The BexBoardroom takes into account the balanceOf of a user who stakes in the rewardPool. This amount is then added to the stakingTokenBalances. Thus this will further incentivize staking in the RewardsPool contract.

**Core Invariants:**

INV 1: If the boardroom is set, accruals on the boardroom must be first updated before staking on the RewardsPool.

**Privileged Functions**

- transferOwnership
- renounceOwnership
- notifyRewardAmount
- recoverERC20
- setRewardsDuration
- setBoardroom

| Issue_20 | Initial staking rewards are left unclaimed |
|---|---|
| **Severity** | **Low** |
| **Description** | The rewards pool is a staking fork of Synthetix and contains its known vulnerability that leaves a portion of the rewards undistributed in the initial stages of staking.<br><br>It occurs due to the notifyRewardAmount function starting off the period with an end determined by the time at which the rewards were added and not when staking actually begins:<br><br>*function notifyRewardAmount(uint256 _amount)*<br>*external*<br>*onlyRole(DEFAULT_ADMIN_ROLE)*<br>*updateReward(address(0)) {*<br>*if (_amount > rewardsToken.balanceOf(address(this))) revert RewardAmountGreaterThanBalance();*<br><br>*if (block.timestamp >= finishAt) {*<br>*rewardRate = _amount / duration;*<br>*} else {*<br>*uint256 remaining = (finishAt - block.timestamp) * rewardRate;*<br>*rewardRate = (_amount + remaining) / duration;*<br>*}*<br><br>*if (rewardRate == 0) revert RewardRateZero();*<br>*if (rewardRate * duration > rewardsToken.balanceOf(address(this))) revert RewardAmountGreaterThanBalance();*<br><br>*finishAt = block.timestamp + duration;*<br>*updatedAt = block.timestamp;*<br><br>*emit NotifyRewardAmount(_amount, finishAt);*<br>*}*<br><br>This leads to the scenario where if we notify rewards at timestamp |

| | |
|---|---|
| | Y, but the first stake occurs at timestamp X, the rewards for the period X - Y will be left undistributed to anyone. These rewards will not be accounted for as remaining by the contract logic and would have to be manually recirculated in the next period.<br>This will occur every time a complete withdrawal of stakes occurs.<br><br>This issue has been rated as low since Synthetix has a recover function |
| **Recommendations** | Consider recovering any lost tokens. |
| **Comments / Resolution** | Acknowledged. |

## BGTRewardsPool

The BGTRewardsPool contract extends upon the functionality of the StakingRewards contract. The main change here is the integration with berachain RewardsVault.

For each stake the contract will mint stUsdbrHoneyLpToken and call delegateStake twice on the rewards vault. The first delegateStake is done for the user/msg.sender, then we will delegateStake on behalf of the owner with the protocol rewards share amount. Finally we call super.stake to call the stake function of the parent contract.

Similarly, for withdrawals the same flow applies. The withdraw function will delegateWithdraw for both the msg.sender and owner. Then we will call the parent function to finish the flow.

**Core Invariants:**

INV 1: A gauge must be set before the contract can function correctly.

**Privileged Functions**

- transferOwnership
- renounceOwnership
- notifyRewardAmount
- recoverERC20
- setRewardsDuration
- setGauge
- approve
- transferTokenOwnership

| Issue_21 | Initial staking rewards are left unclaimed |
|---|---|
| **Severity** | **Low** |
| **Description** | The BGT rewards pool is a staking fork of Synthetix and contains its known vulnerability that leaves a portion of the rewards undistributed in the initial stages of staking.<br><br>It occurs due to the notifyRewardAmount function starting off the period with an end determined by the time at which the rewards were added and not when staking actually begins:<br><br>*function notifyRewardAmount(uint256 _amount)*<br>        *external*<br>        *onlyRole(DEFAULT_ADMIN_ROLE)*<br>        *updateReward(address(0)) {*<br>      *if (_amount > rewardsToken.balanceOf(address(this))) revert RewardAmountGreaterThanBalance();*<br><br>      *if (block.timestamp >= finishAt) {*<br>          *rewardRate = _amount / duration;*<br>      *} else {*<br>          *uint256 remaining = (finishAt - block.timestamp) * rewardRate;*<br>          *rewardRate = (_amount + remaining) / duration;*<br>      *}*<br><br>      *if (rewardRate == 0) revert RewardRateZero();*<br>      *if (rewardRate * duration > rewardsToken.balanceOf(address(this))) revert RewardAmountGreaterThanBalance();*<br><br>      *finishAt = block.timestamp + duration;*<br>      *updatedAt = block.timestamp;*<br><br>      *emit NotifyRewardAmount(_amount, finishAt);*<br>   *}*<br><br>This leads to the scenario where if we notify rewards at timestamp |

| | |
|---|---|
| | Y, but the first stake occurs at timestamp X, the rewards for the period X - Y will be left undistributed to anyone. These rewards will not be accounted for as remaining by the contract logic and would have to be manually recirculated in the next period.<br>This will occur every time a complete withdrawal of stakes occurs.<br><br>This issue has been rated as low since Synthetix has a recover function |
| **Recommendations** | Consider recovering any lost tokens. |
| **Comments / Resolution** | Acknowledged. |

| Issue_22 | Inaccurate comments in BGTRewardsPool |
|---|---|
| **Severity** | **Informational** |
| **Description** | *// 10% (12 / 112)*<br>*uint256 public protocolRewardsShare = 12;*<br><br>from the comments above we can conclude that the protocol intended this value to be 10%, however the protocolRewardsShare is divided by 100 in the actual logic, making the protocolShareReward actually 12% instead of 10%<br><br>*uint256 protocolAmount = amount.mul(protocolRewardsShare).div(100);* |
| **Recommendations** | Consider updating the comments to accurately reflect the percentage fee. |
| **Comments / Resolution** | Acknowledged. |

| Issue_23 | transferTokenOwnership does not set time lock as per comments |
|---|---|
| **Severity** | **Informational** |
| **Description** | The comments states that this emergency function should be under a time lock but it is not enforced in the code<br><br>*// Emergency function, should be under timelock*<br>*function transferTokenOwnership(address _newOwner) public onlyOwner {*<br><br>*IERC20MintBurn(stUsdbrHoneyLpToken).transferOwnership(_newOwner);*<br>*}* |
| **Recommendations** | Consider removing the comment. |
| **Comments / Resolution** | Acknowledged. |

## Oracle

The Oracle contract uses time weighted average price in order to find the average of prices during a specific window.

The recordPrice function allows the operator to record a price. The recordPrice function has the debounce modifier which means it cannot be called again until the debouncePeriod has passed. This makes the oracle resistant to hacks as it will not allow a malicious hijacker to record multiple invalid prices within a short period.

The getTWAP function works by first subtracting the snapshot timestamps from the current timestamp. In this way newer snapshots are thus given more weight compared to snapshots which are older. This is contrary to other twap implementations, for example uni,  which give more weight to older snapshots in order to prevent a large sudden spike in price from greatly affecting the average price.

The getTWAP will only work if 2 or more validSnapshots were recorded within the windowSize. For example if the windowSize is 4 hours and 2 snapshots were recorded, one snapshot 5 hours ago and another snapshot 1 hour ago, then the function will revert because there are not more than 2 validSnapshots.

**Core Invariants:**

INV 1: There must be 2 or more validSnapshots in order for getTWAP to work.

INV 2: snapshots outside of the windowSize will not have any effect on the TWAP.

INV 3: There can only ever be a max of 24 recorded snapshots.

INV 4: A recorded price cannot deviate more from the previous price than the maxDeviation set.

**Privileged Functions**

- transferOwnership
- renounceOwnership

| Issue_24 | Centralization Risk for Operators |
|---|---|
| **Severity** | **Governance** |
| **Description** | A malicious operator can set a random price to grief the protocol. |
| **Recommendations** | Consider incorporating a Gnosis Multisignature contract as owner and ensuring that the Gnosis participants are trusted entities. |
| **Comments / Resolution** | Acknowledged. |

| Issue_25 | oracle is broken in certain scenarios |
|---|---|
| **Severity** | **High** |
| **Description** | The twap accepts snapshots if snapshot.timestamp == curentTime - windowSize as shown below<br><br>*snapshot.timestamp < currentTime.sub(windowSize)) { // Use windowSize variable*<br>*continue;*<br><br>*This is an issue and will cause the twap to malfunction.*<br><br>*validSnapshots++;*<br>*uint256 timeDiff = currentTime.sub(snapshot.timestamp);*<br>*uint256 weight = windowSize.sub(timeDiff);  // Use windowSize variable*<br>*weightedSum = weightedSum.add(snapshot.price.mul(weight));*<br>*totalWeight = totalWeight.add(weight);*<br><br>The problem is that if the current time - windowSize == snapshot.timestamp, it will cause this snapshot to increment validSnapshot while being an invalid timestamp. |

| | |
|---|---|
| | If we see the logic we will see that timeDiff will be equal to window size and that means weight will be 0. The oracle will return only 1 price with weight when in fact 2 or more valid snapshots are needed to ensure a valid twap as can be seen from the logic below.<br><br>*require(validSnapshots >= 2, "Not enough new snapshots available");*<br><br>This results in an invalid snapshot being counted as a validSnapshot and circumvents the check above. This will result in the emissionManager doing rebases with inaccurate information. |
| **Recommendations** | Consider using <= in check.<br><br>*if(snapshot.timestamp == 0 \|\| snapshot.timestamp <= currentTime.sub(windowSize)) { // Use windowSize variable*<br>*continue;* |
| **Comments / Resolution** | Resolved by following recommendations. |

| Issue_26 | getTwap will return wrong price when the synthetic unit is not 1e18 |
|---|---|
| Severity | High |
| Description | In the getTwap function the price is calculator the following way:<br><br>*return weightedSum.div(totalWeight).mul(amount).div(1e18);*<br><br>However this will only return a correct price when the decimals of the synthetic token are 18.<br>For tokens with less decimals the oracle will actually return the wrong price, potentially skipping a rebase. |
| Recommendations | Consider either refactoring the price formula or simply using only tokens with 18 decimals. In the latter scenario, this issue can be safely acknowledged. |
| Comments / Resolution | Acknowledged. |

| Issue_27 | Missing implementation to change the debounce period |
|---|---|
| **Severity** | **Medium** |
| **Description** | Oracle.sol inherits from Debouncable. However it does not implement the internal _setPeriod function, leaving it unreachable:<br><br>*function _setPeriod(uint256 _debouncePeriod) internal {*<br>    *debouncePeriod = _debouncePeriod;*<br>*}*<br><br>This will be problematic when the window size is changed, and the price will need to be updated more often. |
| **Recommendations** | Consider implementing an external setPeriod function in the Oracle contract. |
| **Comments / Resolution** | Acknowledged. |

| Issue_28 | Missing function to remove operator |
|---|---|
| **Severity** | **Low** |
| **Description** | There's no function to remove operators in the Oracle contract, This means that once operators are set, even if they become compromised they can still access important functionalities like recordPrice |
| **Recommendations** | Consider if it makes sense to implement a failsafe function to remove compromised operators. |
| **Comments / Resolution** | Acknowledged. |

| Issue_29 | Price deviation is incorrectly calculated in terms of the latest price |
|---|---|
| Severity | **Low** |
| Description | The Oracle.sol will record prices based on data provided by the trusted operators. There are validations to ensure there aren't large price deviations, however they are done in terms of the last recorded price and not the aimed at peg.<br>This could open up a scenario in which the price continuously increases in acceptable ranges until a rebase is allowed to happen. Since rebases happen every 6 hours and new prices are recorded every 30 minutes, the price could be depegged for an unfavorable period. |
| Recommendations | Consider changing the max deviation to be calculated in terms of the rebasing threshold or the TWAP price. Using the last price allows for continuous price increase in "acceptable" but unfavorable ranges. |
| Comments / Resolution | Acknowledged. |

| Issue_30 | Using the USDbr-Honey pool to determine the price will cause issue in case of a Honey-USD depeg |
|---|---|
| Severity | **Informational** |
| Description | In the documentation it is stated that the USDbr will be pegged to USD, maintaining a 1:1 ratio. However using the USDbr-Honey pool will actually peg the USDbr to the Honey stablecoin. This could be problematic as in the rare event of a Honey-USD depeg, the USDbr token will also depeg from USD. |
| Recommendations | Consider implementing some Honey-USD deviation validation. |
| Comments / Resolution | Acknowledged. |

| Issue_31 | Default maxDeviation is too high |
|---|---|
| **Severity** | **Informational** |
| **Description** | The default maxDeviation for the oracle is set to 200%, this value is too high for a synthetic token expected to maintain a peg with an underlying token(i.e expected price is 1).<br>With a 200% deviation, 0 can be recorded as a valid price as it's only a 100% deviation from peg. |
| **Recommendations** | Consider reducing the maxDeviation, alternatively this issue can be acknowledged. |
| **Comments / Resolution** | Acknowledged. |