yAudit yBGT Review

Review Resources:

- yBGT repo: https://github.com/Bearn-Sucks/yBGT
- bera chain docs: https://docs.berachain.com/

Auditors:

- Watermelon
- Panda

Table of Contents

Review Summary

yBGT

yBGT provides a liquid wrapper for BGT.

The yBGT Repo and two contracts from the yearn tokenized-strategy-periphery codebase were reviewed over six days. Two auditors performed the code review between the 24th and 31st of March 2025. The repository was under active development during the review, but the review was limited to the latest commit at the start. This was commit 5f6584bfd6f0d17a6bc143c0fd2f569c1ce23b26 for the yBGT repo and 7d8559d6e6efd5e9d7d43b7354f9926e2ff9f63b for the yearn tokenized-strategy-periphery codebase.

Scope

The scope of the review consisted of the following contracts at the specific commit:

```
src/
├── BearnAuctionFactory.sol
├── BearnBGT.sol
├── BearnBGTEarnerVault.sol
├── BearnCompoundingVault.sol
├── BearnVaultFactory.sol
├── BearnVoter.sol
├── BearnVoterManager.sol
├── StakedBearnBGT.sol
```

From the yearn tokenized-strategy-periphery codebase, the following contracts were reviewed:

```
src/
├── Bases/
│   ├── Staker/
│   │   └── TokenizedStaker.sol
│   └── Hooks/
│       └── BaseHooks.sol
```

After the findings were presented to the yBGT team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, yBGT and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The codebase implements proper authorization patterns and role-based access control. Privileged functions are protected with appropriate modifiers.
Mathematics	Good	The mathematical operations in the contracts appear sound, with proper consideration for precision and edge cases.
Complexity	Average	While most functions are reasonably straightforward, some components, like the reward distribution and voting management, introduce moderate complexity that requires careful implementation.
Libraries	Good	The project appropriately uses established libraries and components, particularly from the Yearn ecosystem.
Decentralization	Average	The protocol has some centralized components managed by privileged roles but also incorporates decentralized governance mechanisms through BGT voting.
Code stability	Good	The codebase appears stable with well-defined interfaces and consistent implementation patterns.
Documentation	Good	Functions and contracts have appropriate comments explaining their purpose and behavior. The overall architecture is well documented.
Monitoring	Good	The contracts implement sufficient events for tracking important state changes and user interactions.
Testing and verification	Low	Test coverage exists but could benefit from more comprehensive testing of edge cases and integration scenarios.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings

    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

Critical Findings

None

High Findings

1. High - `BearnVoterManager` sets incorrect address as auction's receiver

`BearnVoterManager's constructor` creates an instance of Yearn's `Auction` contract, which is used to create dutch auctions that sell an arbitrary asset, selected by an authorized party, for `HONEY`.

Technical Details

When creating the `Auction` instance, the `BearnVoterManager` provides its own address as the receiver of the auction's proceeds. Given that the contract is only used to manage the `BearnVoter` contract, whose `HONEY` balance is manipulated by the `BearnVoterManager` contract, the auction's receiver address is incorrect.

Because the `BearnVoterManager` contract cannot manipulate its own `HONEY` balance, any auction proceeds will be locked within the contract.

Impact

High. All auction proceeds are directed to the incorrect receiver, locking them perpetually.

Recommendation

Set `address(_bearnVoter)` as the receiver of the auction's proceeds instead of `address(this)`.

```
@@ -60,7 +60,7 @@ contract BearnVoterManager is Authorized {
             AuctionFactory(0xCfA510188884F199fcC6e750764FAAbE6e56ec40)
```

```diff
                  .createNewAuction(
                      address(honey),
-                     address(this),
+                     address(_bearnVoter),
                      styBGT.management(),
                      1 days,
                      5_000
```

Developer Response

Fixed https://github.com/Bearn-Sucks/yBGT/commit/d693c8e4ec2a2bcfa045b171896252a75d7b9671

Medium Findings

1. Medium - `BearnVoterManager.getReward` reads incorrect account's `HONEY` balance

`BearnVoterManager.getReward` is called by `StakedBearnBGT._claimAndNotify` to claim `HONEY` rewards accrued by the `BearnVoter`'s staked `BGT` position.

Technical Details

`BearnVoterManager.getReward` first makes the `BearnVoter` call `BGTStaker.getReward()`, whose current implementation claims the caller's rewards and pushes them to the same account. Because the `HONEY` rewards are pushed to the `BearnVoter` contract, the read at `BearnVoterManager.sol#L261` is incorrectly executed. Subsequently, the `BearnVoter` contract is made to transfer the `HONEY.balanceOf(bearnVoterManager)` amount of `HONEY`.

Three distinct scenarios can arise at this point:

1. `HONEY.balanceOf(bearnVoterManager) == HONEY.balanceOf(bearnVoter)`: In this case, no malfunction is caused by the incorrect balance read.
2. `HONEY.balanceOf(bearnVoterManager) < HONEY.balanceOf(bearnVoter)`: In this case, the `BearnVoter` contract is left holding a non-zero amount of `HONEY`, which in turn implies a lower-than-expected amount of funds being reported as profit to `StakedBearnBGT`, lowering its reward rate.
3. `HONEY.balanceOf(bearnVoterManager) > HONEY.balanceOf(bearnVoter)`: In this case, the `HONEY` transfer will revert as the `BearnVoter` contract holds insufficient funds to

execute such transfer.

Case 3 is the worst case: it implies a partial DoS on `StakedBearnBGT`'s reward claim flow, which will be resolved once the `BearnVoter` instance has accrued sufficient rewards for its balance to surpass that of `BearnVoterManager`.

Impact

Medium.

Recommendation

Execute the balance read correctly by reading the `BearnVault`'s balance.

```
@@ -258,7 +258,7 @@ contract BearnVoterManager is Authorized {
         );

         // Transfer Full balance of honey to styBGT to account for auctions.
-        uint256 amount = honey.balanceOf(address(this));
+        uint256 amount = honey.balanceOf(address(bearnVoter));

         // Send rewards to styBGT
         if (amount > 0) {
```

Developer Response

Fixed https://github.com/Bearn-Sucks/yBGT/commit/d693c8e4ec2a2bcfa045b171896252a75d7b9671

2. Medium - Users of BearnBGT can't signal intention of withdraw to bera

Currently, BearnBGT (yBGT) users face a significant limitation: they are locked into the protocol without a reliable way to signal their intention to withdraw or exit when their BGT tokens are being used for voting. The existing `redeem()` function only allows the withdrawal of unboosted (non-voting) BGT tokens. This means users can be effectively trapped in the protocol if most or all of their yBGT is used for governance voting.

This creates several problems:

1. Users cannot reliably exit their positions
2. The protocol lacks transparency on withdrawal demand

3. There's no mechanism to prioritize and fulfill withdrawal requests as tokens become available
4. Voter operators have no visibility into user withdrawal intentions

Technical Details

The current implementation in `BearnBGT.sol` has a `redeem()` function that checks the available unboosted balance and only allows withdrawal up to that amount:

```
function maxRedeem() public view returns (uint256 maxAmount) {
    return IBGT(bearnVoter.bgt()).unboostedBalanceOf(address(bearnVoter));
}

function redeem(uint256 amount) external returns (uint256 outputAmount) {
    uint256 fee;
    (outputAmount, fee) = feeModule.redeem(msg.sender, amount);

    // burn and redeem to msg.sender
    if (outputAmount > 0) {
        _burn(msg.sender, outputAmount);
        bearnVoter.redeem(msg.sender, outputAmount);
    }

    // transfer fees to fee recipient
    if (fee > 0) {
        _transfer(msg.sender, treasury, fee);
    }

    return outputAmount;
}
```

This means that if all tokens are being used for voting (boosted), users cannot withdraw them.

Impact

Medium. Decreases trust in the protocol due to uncertainty about liquidity

Recommendation

Implement a Withdrawal Queue similar to Lido's approach with the following components:

1. Withdrawal Request Queue

- Allow users to enter a queue for withdrawals that exceed current unboosted balance
- Users specify the amount to withdraw and receive a withdrawal NFT/receipt as proof
- This signals intent without immediate execution

2. Withdrawal Fulfillment Logic

- Newly emitted BGT should first fulfill pending withdrawal requests before being used for voting
- The vote operator should be able to see pending withdrawal requests and their total amount
- Implement logic to gradually unboost to fulfill withdrawal requests over time

Developer Response

Acknowledged

As of now, users are expected to exit yBGT through the AMM pools. We anticipate yBGT trading at a premium for a while, meaning redemptions should not be used.

The logic for the withdrawal queue can be added to the current system, and we plan to implement this functionality in the future.

Even if redemptions become necessary now, reports are also permissionless. This is when BGT is claimed and becomes unboosted, so redemptions could be paired with permissionless reports to free up unboosted BGT.

Low Findings

1. Low - Unbounded `rewardTokens` array in `addReward` can lead to DOS

Technical Details

The addReward function in the TokenizedStaker contract allows the management role to add new reward tokens to the contract.
There’s no limit to how many reward tokens can be added, and the `_updateReward` function iterates through all tokens in the rewardTokens array:

```
File: TokenizedStaker.sol
362:         for (uint256 i; i < rewardTokens.length; ++i) {
363:             address _rewardToken = rewardTokens[i];
```

Impact

Low.

Recommendation

Add an upper limit to the number of tokens in `rewardTokens`.

Developer Response

Acknowledged

2. Low - Inconsistent default behaviour in `BaseHooks` withdraw methods

`BaseHooks.withdraw` and `BaseHooks.redeem` provide overloaded methods which define different `maxLoss` hardcoded values.

Technical Details

While `BaseHooks.withdraw` inserts a `maxLoss = 0` to disallow any potential loss to the user within the withdrawal process, `BaseHooks.redeem` inserts `maxLoss = MAX_BPS`, effectively disabling the user loss check completely.

Impact

Low. Enabling a strict loss check on `BaseHooks.withdraw` by default might mislead contract users into believing the same is true for `BaseHooks.redeem`, exposing them to a potential 100% loss of funds.

Recommendation

Maintain consistent behavior across both methods: ideally, both methods should employ `maxLoss = 0`.

Developer Response

Acknowledged

This is done to match the same default functionality as the TokenizedStrategy itself has. Which is dont to match the 4626 standard. Since 'withdraw' must return the exact amount of asset requested (0 max loss), but redeem must just burn the amount of shares.

We allow loss on the redeem calls to make downstream integrations easier specifically when staking 4626 calls in order to not revert from simple rounding losses of a few wei

Gas Saving Findings

1. Gas - `recoverERC20` can disrupt reward token distribution

Technical Details

The `recoverERC20` function allows management to recover tokens from the contract, with a safeguard to prevent recovering reward tokens until 90 days after the reward period ends:

```
function recoverERC20(
    address _tokenAddress,
    uint256 _tokenAmount
) external virtual onlyManagement {
    // Logic to check if it's a reward token and enforce 90-day wait
    if (isRewardToken) {
        require(
            block.timestamp > maxPeriodFinish + 90 days,
            "wait >90 days"
        );
        // Takes all of the token balance
        _tokenAmount = ERC20(_tokenAddress).balanceOf(address(this));
    }
    // Transfer tokens to management
}
```

The issue is that even after recovering a reward token, it remains in the `rewardTokens` array and continues to be processed during reward updates and claims. This creates several inefficiencies:

1. Failed token transfers may occur in `_getRewardFor` if all of a reward token was recovered
2. Gas is wasted on processing tokens that can no longer be distributed
3. The contract maintains reward accounting for tokens that have been removed

Impact

Gas savings.

Recommendation

1. Add a `deactivated` flag to the `Reward` struct
2. Set this flag when recovering all of a reward token
3. Modify `_updateReward` and `_getRewardFor` to skip deactivated tokens

Example implementation:

```
// Add to Reward struct
bool deactivated;

// In recoverERC20, when taking all tokens
if (_tokenAmount == ERC20(_tokenAddress).balanceOf(address(this))) {
    rewardData[_tokenAddress].deactivated = true;
}

// In _updateReward and _getRewardFor
if (rewardData[rewardToken].deactivated) continue;
```

Developer Response

Acknowledged

2. Gas - `BearnVoteManager.activateBoost` and `BearnVoteManager.dropBoost` route external calls to `BearnVoter` needlessly

`BearnVoteManager.activateBoost` and `BearnVoteManager.dropBoost` are methods which implement no access control and simply execute an external call to `BearnVoter.execute`, making the contract execute an external call to Berachain's `BGT` token.

Technical Details

Because `BGT.activateBoost` and `BGT.dropBoost` do not require for the holder of a boost to be the calling account, any account is allowed to activate and drop a boost on behalf of another account if enough time has passed since the boost activation or deactivation has been requested.

It is thus unnecessary for `BearnVoterManager` to route such calls via `BearnVoter`, when it is allowed to call into `BGT` directly.

Impact

Gas savings.

Recommendation

Modify `BearnVoteManager.activateBoost` and `BearnVoteManager.dropBoost` to interact directly with `BGT`, invoking the corresponding methods, taking care to specify the `BearnVoter` contract as the `user` parameter for such calls.

Developer Response

Fixed https://github.com/Bearn-Sucks/yBGT/commit/d693c8e4ec2a2bcfa045b171896252a75d7b9671

Informational Findings

1. Informational - `TokenizedStaker` misses a method for an authorized party to claim a single reward on a user's behalf

`TokenizedStaker` implements logic for a user to approve another account to claim and receive its rewards via the `setClaimForSelf` method.

Technical Details

While the authorized account can claim all of the user's rewards by calling `getRewardFor`, it doesn't dispose of a method to claim a single reward, similarly to how the rewards owner can do by invoking `getOneReward`.

Impact

Informational.

Recommendation

Implement a `getOneRewardFor` method to offer authorized accounts the same functionality, allowing them to claim only one reward token on a user's behalf.

Developer Response

Acknowledged

2. Informational - Missing event logs

Technical Details

1. When modifying the `TokenizedStaker.claimForRecipient` mapping via `TokenizedStaker.setClaimFor` or `TokenizedStaker.setClaimForSelf`, no event log is emitted to signal the storage modification to off-chain components.
2. When adding a reward via `TokenizedReward.addReward`, no event is emitted.

Impact

Informational.

Recommendation

1. Define a `ClaimSetFor(address staker, address recipient)` event and log it at the end of `_setClaimFor`.
2. Define a `RewardTokenAdded(address _rewardToken, address _rewardsDistributor, uint256 _rewardsDuration)` event and log it at the end of `_addReward`.

Developer Response

Acknowledged

Final remarks

The yBGT codebase demonstrates solid engineering practices with good access control and documentation.
However, fixes are required for the identified issues related to reward handling and withdrawal mechanisms to ensure proper functionality and user experience.