SB SECURITY

BeeBribes

Security Review

March 31, 2025

Contents

1. About SBSecurity

SBSecurity is a duo of skilled smart contract security researchers. Based on the audits conducted and numerous vulnerabilities reported, we strive to provide the absolute best security service and client satisfaction. While it's understood that 100% security and bug-free code cannot be guaranteed by anyone, we are committed to giving our utmost to provide the best possible outcome for you and your product.

Book a Security Review with us at sbsecurity.net or reach out on Twitter @Slavcheww.

2. Disclaimer

A smart contract security review can only show the presence of vulnerabilities **but not their absence**. Audits are a time, resource, and expertise-bound effort where skilled technicians evaluate the codebase and their dependencies using various techniques to find as many flaws as possible and suggest security-related improvements. We as a company stand behind our brand and the level of service that is provided but also recommend subsequent security reviews, on-chain monitoring, and high whitehat incentivization.

3. Risk classification

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

3.1. Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - leads to a moderate loss of assets in the protocol or some disruption of the protocol's functionality.
- **Low** - funds are not at risk.

3.2. Likelihood

- **High** - almost **certain** to happen, easy to perform, or highly incentivized.
- **Medium** - only **conditionally possible**, but still relatively likely.
- **Low** - requires specific state or **little-to-no incentive**.

3.3. Action required for severity levels

- High - **Must** fix (before deployment if not already deployed).
- Medium - **Should** fix.
- Low - **Could** fix.

4. Executive Summary

BeeBribes contracts have been audited through the Hyacinth platform.

Overview

Project	BeeBribes
Repository	Private
Commit Hash	19a1694eb7be8d0e9e713c9c436c1248a4302dd1
Resolution	20bad9a5c172e7e3a95dcb9282e4ad52a3629472
Timeline	March 4, 2025 - March 6, 2025 March 16, 2025 - March 18, 2025

Scope

BeeBribeswrapper.sol
BeeBribesRewards.sol
BeeBribesOperator.sol - distributeBgtRewards()

Issues Found

Critical Risk	0
High Risk	3
Medium Risk	3
Low/Info Risk	6

5. Findings

5.1. High severity

5.1.1. Beacon deposit won't work

Severity: High Risk

Description: When making beacon deposits, the `BeaconDeposit` contract has several restrictions on the amount that can be deposited:

- It must be multiple of `1e9 (gwei)`

- `amount / 1e9` must be less than `uint64.max`

```
/// @dev Validates the deposit amount and sends the native asset to the zero address.
function _deposit() internal virtual returns (uint64) {
    if (msg.value % 1 gwei != 0) {
        DepositNotMultipleOfGwei.selector.revertWith();
    }

    uint256 amountInGwei = msg.value / 1 gwei;
    if (amountInGwei > type(uint64).max) {
        DepositValueTooHigh.selector.revertWith();
    }

    _safeTransferETH(address(0), msg.value);

    return uint64(amountInGwei);
}
```

Recommendation: Inside `_depositOnBeacon()`, divide `pendingBalance` by 1e9 and then multiply the result by 1e9 to find the value that can be passed to `BEACON_DEPOSIT.deposit()`. Also add a check to see if the result of the division is greater than `uint64.max`.

Another suggestion is to hardcode the deposited amount, this is the easiest and most error-prone implementation, it will also remove the possibility of deposit to fail on a Consensus level due to wrong signature provided.

Resolution: Fixed

5.1.2. If validator is exited, next deposit batch is lost

Severity: High Risk

Description: Validator can become exited, if there are 69 validators with more staked BERA, then all his staked tokens are refunded, since there's no slashing in Berachain yet and no further deposits are accepted.

But there's no onchain mechanism to stop the deposits. As a result, if `BeeBribesWrapper` triggers a deposit to the `BeaconDeposit` contract, all the tokens will be burned forever. This is because the `pubkey` is no longer valid, as explained here.

Recommendation: On a smart contract level - add a pausing mechanism. On the infrastructure level - consider adding an offchain trigger/bot that can react when the Validator is being exited and prevent the `BeeBribesWrapper::_depositOnBeacon` from executing. You will have ~1 epoch to cut the ability for deposit.

Resolution: Fixed

5.1.3. Dynamic balance beacon deposit tokens will be lost due to a wrong signature

Severity: High Risk

Description: Bera amount that is being deposited into Beacon is dynamic, which will make almost all the tokens lost because the `signature` argument must be calculated based on the amount forwarded. This is not possible now, since all the balance of `BeeBribesWrapper` is deployed always.

Here is what Berachain docs say:

"Ensure you calculate a signature for the parameters and deposit amount you are going to send to the deposit contract. If you change anything after calculating the signature, the deposit will fail and the funds will be burnt."

For example, let's say the signature is calculated for 10k Bera tokens, but `depositOnBeacon` is invoked on 10k + 1, then all the tokens are lost. The only way that tokens will be properly deposited the balance must be exactly equal to 10k.

Recommendation: Hardcode the deposit amount to 10k, then modify the logic, to trigger auto-deposit to beacon whenever the balance is at least equal to the deposit amount. Knowing the numbers in advance will allow you to calculate the proper signature.

Resolution: Fixed

5.2. Medium severity

5.2.1. Frontrunning the initial beacon deposit will block BeeBribesWrapper

Severity: Medium Risk

Description: Adversary can frontrun the initial `BeaconDeposit` transaction by setting his own operator, as a result, the `BeeBribesWrapper` will be temporarily blocked and the new validator will have to be created. The reason for that is the fact that the `BeaconDeposit` contract has a "frontrun" protection, that requires passing a non-zero operator only for the first deposit for a given `pubkey`:

```
function deposit(
    bytes calldata pubkey,
    bytes calldata credentials,
    bytes calldata signature,
    address operator
)
    external
    payable
{
   ...MORE CODE
    // Set operator on the first deposit.
    // zero `_operatorByPubKey[pubkey]` means the pubkey is not registered.
    if (_operatorByPubKey[pubkey] == address(0)) {
        if (operator == address(0)) {
            ZeroOperatorOnFirstDeposit.selector.revertWith();
        }
        _operatorByPubKey[pubkey] = operator;
        emit OperatorUpdated(pubkey, operator, address(0));
    }
    // If not the first deposit, the operator's address must be 0.
    // This prevents from the front-running of the first deposit to set the operator.
    else if (operator != address(0)) {
        OperatorAlreadySet.selector.revertWith();
    }
...MORE CODE
}
```

The only thing stopping the attacker is the fact he should sacrifice at least 10k Bera tokens.

Recommendation: The current design leaves room for changing the validator credential and retrying the initial deposit with the proper operator. One suggestion is to make the first deposit admin-controlled and then batch the `setValidatorInformation` with the `depositOnBeacon` through a private mempool.

Resolution: Fixed

5.2.2. `distributeWBera` will run out of gas when stakers grow

Severity: Medium Risk

Description: There's no limit on the max active BERA token stakers, also there's no functionality to remove from the `stakers` array. As a result, **when** the array grows reasonably big in length (worst-case 1 million stakers, each with 100 BEAR deposited, max 10 million per Validator), all the reward distribution will be blocked without a way to be fixed. This is because it always iterates over the stakers and calculates their relative BGT rewards.

```
function distributeWBera(uint256 amount) external onlyRole(OPERATOR) {
    if (WBERA.allowance(msg.sender, address(this)) < amount)
        revert InvalidAllowance();

    if (stakers.length() == 0)
        revert NoStaker();

    address wberaAddress = address(WBERA);
    uint256 totalDistributed;

    for (uint256 i; i < stakers.length(); ++i) {
        (address staker, uint256 balance) = stakers.at(i);
        uint256 weight = ONE_HUNDRED_PERCENT.mulDiv(balance, totalBeraAmount);
        uint256 share = weight.mulDiv(amount, ONE_HUNDRED_PERCENT);
        beeBribesRewards.distributeReward(staker, wberaAddress, share);
        totalDistributed += share;
    }
    // Use total distributed, because we can have so loss in the precision so we only want to distribute
the right amount and left the dust behind.
    WBERA.transferFrom(msg.sender, address(beeBribesRewards), totalDistributed);
}
```

Recommendation: Some possible mitigations of this issue:

1. Introduce max stakers limit, with an ordered array by the biggest stakers. The downside is that you will limit the participation.

2. Think of a simplified Synthetix-like reward distribution mechanism. Track the reward index and calculate the rewards due based on the index, instead of iterating over the stakers each time.

Resolution: Acknowledged

5.2.3. Refunding full staker amount doesn't remove him from the map

Severity: Medium Risk

Description: In addition to M-02, when staker is refunded **completely**, he is no deleted from the `stakers` map. This will also contribute to the size growth and lead to distribution running out of gas.

```
function refundStaker(address staker, uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
    (bool exists, uint256 value) = stakers.tryGet(staker);
    uint256 balance = address(this).balance;

    if(!exists)
        revert StakerNotFound();

    if (amount > value)
        revert InvalidAmount();

    if (amount > balance)
        revert InvalidAmount();

    EnumerableMap.set(stakers, staker, value - amount);//NOTE: not removed from the map when value ==
amount
    totalBeraAmount -= amount;
    (bool sent,) = payable(staker).call{value: amount}("");
    if(!sent)
        revert TransferFailed();
}
```

Recommendation:

```
function refundStaker(address staker, uint256 amount) external onlyRole(DEFAULT_ADMIN_ROLE) {
        (bool exists, uint256 value) = stakers.tryGet(staker);
        uint256 balance = address(this).balance;

        if(!exists)
            revert StakerNotFound();

        if (amount > value)
            revert InvalidAmount();

        if (amount > balance)
            revert InvalidAmount();
+       if(value == amount) {
+          EnumerableMap.remove(stakers, staker);
+       }
+       else {
           EnumerableMap.set(stakers, staker, value - amount);
+       }
        totalBeraAmount -= amount;
        (bool sent,) = payable(staker).call{value: amount}("");
        if(!sent)
            revert TransferFailed();
    }
```

Resolution: Fixed

5.3. Low severity

5.3.1. Replicate beacon contract deposit validator checks in `BeeBribesWrapper::setValidatorInformation`

Severity: Low Risk

Description: In `BeaconDeposit.sol` there are checks for the `pubkey`, `credentials` and `signature` length.

```
function deposit(
    bytes calldata pubkey,
    bytes calldata credentials,
    bytes calldata signature,
    address operator
)
    external
    payable
{
    if (pubkey.length != PUBLIC_KEY_LENGTH) {
        InvalidPubKeyLength.selector.revertWith();
    }

    if (credentials.length != CREDENTIALS_LENGTH) {
        InvalidCredentialsLength.selector.revertWith();
    }

    if (signature.length != SIGNATURE_LENGTH) {
        InvalidSignatureLength.selector.revertWith();
    }
```

Recommendation: Make sure to add them in `setValidatorInformation()` also.

Resolution: Fixed

5.3.2. In `BeeBribesWrapper::deposit` use `msg.value` as amount

Severity: Low Risk

Description: In `BeeBribesWrapper::deposit` the amount argument can be omitted because the whole functionality relies on the `msg.value`. That way refund won't be needed, which will simplify the functionality and will remove the reentrancy possibility.

Recommendation:

```
- function deposit(uint256 amount) external payable {
+ function deposit() external payable {
        if(msg.value < minBeraAmount) revert InvalidAmount();
-       if(msg.value < amount) revert InvalidAmount();
+       uint256 amount = msg.value;

        (bool e, uint256 value) = pendingStakers.tryGet(msg.sender);

        if (e)
            pendingStakers.set(msg.sender, amount + value);
        else
            pendingStakers.set(msg.sender, amount);

        if (amount >= thresholdAmount && validatorRegistered)
            _depositOnBeacon();

-       if (msg.value > amount) {
-           // refund if too much bera has been sent
-           (bool sent,) = payable(msg.sender).call{value: msg.value - amount}("");
-           if(!sent)
-               revert TransferFailed();
-       }
    }

function _depositOnBeacon() internal {
        ...MORE CODE
-       if(pendingBalance > balance)
-           revert InvalidBeraBalance(balance, pendingBalance);

        if(!validatorRegistered) {
            BEACON_DEPOSIT.deposit{value: pendingBalance}(
                pubkey,
                credential,
                signature,
                operator
            ); // Send bera (native tokens)
            validatorRegistered = true;
        } else {
            BEACON_DEPOSIT.deposit{value: pendingBalance}(
                pubkey,
                credential,
                signature,
                address(0)
            );
        }
    }
```

Resolution: Fixed

5.3.3. Leftover allowance after distributeWBera

Severity: Low Risk

Description: There will be accumulating allowance in `BeeBribesOperator::distributeBgtRewards`, in particular the `BeeBribesWrapper::distributeWBera` due to the division truncations. Since `WBERA` doesn't have issues with leftover approvals and the contract that receives the approval is trusted, this is not a security vulnerability but more of a good practice to zero out after the operation.

Recommendation: Either clear the approval or simply give infinite approval (`type(uint256).max`) in the `constructor` and `setBeeBribesWrapper`, since `WBERA` supports it.

Resolution: Fixed

5.3.4. `Wrapper::refundStaker` cannot be used

Severity: Low Risk

Description: `Wrapper::refundStaker` cannot be used because `totalBeraAmount` doesn't indicate the `BERA` balance of the Wrapper but the staked amount on behalf of the Validator.

That means no refund should happen, otherwise, this will take from the funds of the pending stakers.

Recommendation: Remove the functionality to refund the stakers, for the scenario when the validator is exited and the staked tokens should be refunded to the original stakers, it would be better to code a separate contract that can handle the distribution.

Resolution: Fixed

5.3.5. `BeeBribesOperator` is missing functionality to change operator

Severity: Low Risk

Description: There should be functionality in the `BeeBribesOperator` that can propose a new operator for the Lavender validator in case a change/upgrade is needed.

Recommendation: In `BeeBribesOperator` add an access-controlled function to call the `BeaconDeposit::requestOperatorChange` function.

Resolution: Fixed

5.3.6. Setter functions are missing bound checks

Severity: Low Risk

Description: `setErc20Commission` and `setBgtCommission` in `BeeBribesOperator` should have checks preventing from giving bigger number than `ONE_HUNDRED_PERCENT`. Otherwise, there's a chance the admin to pass wrong amount which can temporary block the reward distributions.

Recommendation:

```
function setBgtCommission(uint128 _newCommission) external onlyRole(DEFAULT_ADMIN_ROLE) {
+       require(_newCommission <= ONE_HUNDRED_PERCENT, "Cannot exceed 100%");
        bgtCommission = _newCommission;
    }

function setErc20Commission(uint128 _newCommission) external onlyRole(DEFAULT_ADMIN_ROLE) {
+       require(_newCommission <= ONE_HUNDRED_PERCENT, "Cannot exceed 100%");
        erc20Commission = _newCommission;
    }
```

Resolution: Fixed

5.4. Centralization/Governance severity

5.4.1. Centralization

Severity: Centralization Risk

Description: There is no mechanism to refund stakers if the validator is withdrawn or exited. All the BERA staked by the users will be sent to a single address, which is chosen by the admins. That lowers the decentralization since these funds **must be manually** sent to the `BeeBribesWrapper`, from where users can claim.

Recommendation: A Separate contract can be coded to be used as a withdrawal bank, when Voluntary Withdrawals are enabled.

Resolution: Acknowledged