yAudit Beefy Sonic Review

Review Resources:

- None beyond the code repositories

Auditors:

- fedebianu
- HHK

Table of Contents

Review Summary

Beefy Sonic

Beefy enables liquid staking of Sonic (S) tokens on the Sonic chain. The protocol allows users to stake their S tokens and receive beS (Beefy Sonic) tokens in return, representing their staked position while maintaining liquidity. Users can trade or use these beS tokens as every LST while earning staking rewards from their underlying S tokens.

The contracts of the Beefy Sonic Repo were reviewed over 3 days. Two auditors performed the code review between the 24th and 26th of March 2025. The review was limited to the latest commit ed087129fb81f38c66b323ecd4a01f430bb97a8c for the Beefy Sonic repo.

Scope

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

```
├── BeefySonic.sol
├── BeefySonicStorageUtils.sol
└── interfaces
    ├── IBeefySonic.sol
    ├── IConstantsManager.sol
    ├── IFeeConfig.sol
```

```
    ├── ISFC.sol
    └── IWrappedNative.sol
```

After the findings were presented to the Beefy 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, Beefy and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Well-implemented role system with owner, keeper and ERC-7540 controller and operator. Critical functions have appropriate access controls.
Mathematics	Good	No complex math operations. Proper handling of decimals and precision (1e18). Safe operations for fee calculations and share/asset conversions.
Complexity	Good	Generally clean code structure. Some complexity is added by slashing management, but it's well handled through dedicated functions.
Libraries	Good	Good use of OpenZeppelin's battle-tested contracts (ERC4626, Upgradeable patterns, SafeERC20).
Decentralization	Medium	While staking is permissionless, significant power remains with admin roles.
Code stability	Good	Proper use of UUPS upgrade pattern. Well-structured storage layout. Clear separation of concerns.
Documentation	Low	Minimal inline documentation. Complex mechanisms (e.g., slashing management) need better documentation. Missing architectural overview and detailed specifications.
Monitoring	Good	Key events are emitted.
Testing and verification	Medium	High test coverage, but it lacks fuzzing and invariant testing.

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.

Medium Findings

1. Medium - `harvest()` can be gamed to gain previously accumulated rewards

The harvest mechanism combines reward claiming and delegation into a single atomic operation. This design can dilute rewards when validator capacity is full, as new users can back-run validator additions to capture accumulated rewards.

Technical Details

`harvest()` combines two operations:

1. Claiming rewards
2. Delegating them into validators

This creates a scenario where:

1. If all validators are at capacity, rewards accumulate but can't be harvested
2. A new validator needs to be added by the owner that is a timelock
3. When the new validator is added, anyone can back-run this transaction by:
    - Depositing right after the new validator is added
    - Harvesting immediately after their deposit
    - Gaining a share of all accumulated rewards

Impact

Medium. Users who held shares during the accumulation period are diluted by new users who enter after the new validator capacity becomes available.

Recommendation

Separate the harvest flow into two distinct functions:

```
function harvestRewards() external whenNotPaused {
    BeefySonicStorage storage $ = getBeefySonicStorage();

    // We just revert if the last harvest was within the lock duration to prevent
ddos
    if (block.timestamp - $.lastHarvest <= $.lockDuration) revert
NotReadyForHarvest();

    // Claim pending rewards
    uint256 beforeBal = address(this).balance;
    _claim();
    uint256 claimed = address(this).balance - beforeBal;
    emit ClaimedRewards(claimed);

    // Check if there is enough rewards
    if (claimed < $.minHarvest) revert NotEnoughRewards();

    // Charge fees
    _chargeFees(claimed);

    // Balance of Native on the contract this includes Sonic after fees and
donations
    // You can technically donate by calling withdrawTo with to being this
address on wS
    uint256 contractBalance = address(this).balance;

    // Update stored total and total locked
    $.totalLocked = lockedProfit() + contractBalance;
    $.storedTotal += contractBalance;
    $.lastHarvest = block.timestamp;

    emit RewardsClaimed(claimed);

    // Try to delegate rewards, but don't revert if no validator capacity
    try delegateRewards() {
        // Success
    } catch {
        // Delegation failed (likely no validator capacity)
        // Rewards will stay in contract until capacity is available
        emit RewardsDelegationFailed(contractBalance);
    }
}
```

```
function delegateRewards() public whenNotPaused {
    BeefySonicStorage storage $ = getBeefySonicStorage();

    // Balance of Native on the contract this includes Sonic after fees and
donations
    uint256 contractBalance = address(this).balance;
    if (contractBalance == 0) revert NoRewardsToDeploy();

    // Get validator to deposit
    uint256 validatorId = _getValidatorToDeposit(contractBalance);

    // Get validator from storage
    Validator storage validator = $.validators[validatorId];

    // Update delegations
    validator.delegations += contractBalance;

    // Delegate assets to the validator
    ISFC($.stakingContract).delegate{value: contractBalance}(validator.id);

    emit RewardsDelegated(contractBalance, validatorId);
}
```

This separation allows:

1. Rewards to be harvested even when validators are full
2. The possibility to delegate donations at any time
3. Users will benefit from accumulated rewards if they want to withdraw
4. You can choose to use these free funds as a buffer in the withdrawal process

Additionally, consider refactoring the withdrawal process to use the unstaked `S` tokens.

Developer Response

Fixed in PR#28.

A temporary edge-case remain if most users start withdrawing as the share value takes into account `undelegatedShares` but those shares are not yet redeemable. Some users may have to wait for these funds to be delegated to fully withdraw.

2. Medium - New users can deposit while slashing is being processed

When a validator is slashed, the Beefy admins need to process it to socialize the loss. However, while the admins are taking care of this, deposits are still open if another validator is available. New users may see their deposits reduced even if they deposited after the slashing.

Technical Details

The function `_deposit()` will mint shares and deposit them into a valid validator, reverting if none are found.

However, if multiple validators are present, with one of them being slashed, the deposit will still proceed as it is able to find a validator that wasn't slashed.

When the beefy admins socialize the loss by calling `checkForSlashedValidatorsAndUndelegate()` and `completeSlashedValidatorWithdraw()` the users that deposited after the slash will see their share value lowered.

Impact

Medium. Users unaware of a validator being slashed will be part of the socialization even though they deposited after the event.

Recommendation

Revert deposits if a validator has been slashed and hasn't been dealt with or rework how the contract deals with slashing to socialize the loss and not require admins actions instantly.

Developer Response

Fixed in PR#33.

3. Medium - Users can exit at full share value while slashing is being socialized

When socializing, users shouldn't be able to leave the vault unless they accept to take on the loss; however, it is currently possible for a user to exit the vault with the full share value.

Technical Details

When the Beefy team starts the socialization process, they call `checkForSlashedValidatorsAndUndelegate()`, which is going to set `validator.delegations == 0` and call `SFC.undelegate()` if the refund ratio is greater than zero.

Later, after waiting for the `withdrawDuration()`, they can call `completeSlashedValidatorWithdraw()` to socialize the loss.

However, in the case of multiple validators and only one slashed, a user is still able to call `_requestRedeem()`.
This is because by setting the `validator.delegations == 0` the `_getValidatorsToWithdraw()` will ignore the slashed validator and continue inside the loop: `if (delegations == 0) continue;`.

Because the `checkForSlashedValidatorsAndUndelegate()` doesn't reduce the `storedTotal`, the share value hasn't been decreased yet. So users can make requests at the full share value before the beefy admins call `completeSlashedValidatorWithdraw()` as long as other valid validators have enough funds.

Impact

Medium. Users monitoring slashing events will be able to bypass socialization.

Recommendation

Check if socialization is ongoing for a validator before continuing the loop and revert if that's the case.

Developer Response

Fixed in PR#38.

Low Findings

1. Low - Theoretical DoS in `harvest()` due to unbounded validators loop

Technical Details

`_claim()`, called by `harvest()`, loops through all validators without any limit:

1. Each validator requires external calls (`pendingRewards()`, `claimRewards()`, `getValidator()` in `_validatorStatus()`)

2. The number of validators can grow indefinitely through `addValidator()`

```
function _claim() private {
    BeefySonicStorage storage $ = getBeefySonicStorage();
    for (uint256 i; i < $.validators.length; ++i) {  // Unbounded loop
        if (validator.claim) {
            // 3 external calls per active validator
            uint256 pending =
ISFC($.stakingContract).pendingRewards(address(this), validator.id);
            if (pending > 0) ISFC($.stakingContract).claimRewards(validator.id);
            (bool isOk,) = _validatorStatus(validator.id);
            // ...
        }
    }
}
```

If too many validators are added, **`harvest()`** could exceed the block gas limit, making rewards impossible to claim.

However, as the Sonic block gas limit is currently 5B, it is highly unlikely that a transaction would reach that number, as the function would need to loop over ~20,000 validators.

Impact

Low.

Recommendation

Limit the maximum number of validators or implement a batched claim mechanism:

```
function harvest() external whenNotPaused {
    BeefySonicStorage storage $ = getBeefySonicStorage();
    harvest(0, $.validators.length - 1);
}

function harvest(uint256 startIndex, uint256 endIndex) public whenNotPaused {
    // ...
    _claim(startIndex, endIndex);
    // ...
}

function _claim(uint256 startIndex, uint256 endIndex) private {
```

```
    BeefySonicStorage storage $ = getBeefySonicStorage();
    if (endIndex > $.validators.length) revert InvalidIndex();

    for (uint256 i = startIndex; i <= endIndex; ++i) {
        Validator storage validator = $.validators[i];
        if (validator.claim) {
            // ...
        }
    }
}
```

Developer Response

Acknowledged. We probably won't add this just due to contract size constraints and the fact that its very unlikely an array can get to a size where this is an issue.

Gas Saving Findings

1. Gas - Fail earlier

Technical Details

`setLiquidityFee()` performs validation after storage operations and other logic, which wastes gas if the validation fails.

- L1024-L1027: `setLiquidityFee`

```
function setLiquidityFee(uint256 _liquidityFee) external onlyOwner {
    BeefySonicStorage storage $ = getBeefySonicStorage();
    emit LiquidityFeeSet($.liquidityFee, _liquidityFee);
    if (_liquidityFee > 0.1e18) revert InvalidLiquidityFee();
    $.liquidityFee = _liquidityFee;
}
```

Impact

Gas savings.

Recommendation

Move validations to the beginning of functions:

```
function setLiquidityFee(uint256 _liquidityFee) external onlyOwner {
    if (_liquidityFee > 0.1e18) revert InvalidLiquidityFee();
    BeefySonicStorage storage $ = getBeefySonicStorage();
    uint256 oldValue = $.liquidityFee;
    $.liquidityFee = _liquidityFee;
    emit LiquidityFeeSet(oldValue, _liquidityFee);
}
```

Developer Response

Acknowledged.

2. Gas - Inverse check order in `_onlyOperatorOrController()`

Technical Details

In the modifier `_onlyOperatorOrController()`, the condition can be modified to first check `_controller != msg.sender` as the `controller` will most likely always be the `msg.sender`, and it will save a storage read.

Impact

Gas.

Recommendation

Invert the checks in the condition.

Developer Response

Fixed in PR#32.

3. Gas - Cache storage variables

Technical Details

Throughout the contract, storage variables are sometimes accessed multiple times inside the same function. Consider caching them to save storage read.

- In `_claim()` `validator.id` is used multiple times
- In `_getValidatorToDeposit()` `validators.length` is used multiple times.
- In `_requestRedeem()` `$.wId` might be read and updated multiple times in the loop.
- In `completeSlashedValidatorWithdraw()` `stakingContract` is used multiple times.

- In `_removeRequest()` `pendingRequests.length` is used multiple times.
- In `_chargeFees()` `liquidityFee` is used multiple times.
- In `addValidator()` `validators.length` is used multiple times.

Impact

Gas.

Recommendation

Cache the storage variable in memory.

Developer Response

Acknowledged.

4. Gas - `lockedProfit()` not needed inside `harvest()`

Technical Details

Inside the function `harvest()`, there is a call to `lockedProfit()`; however, that call is not needed because the function can't be called if it's been less than `lockDuration` since the last harvesting. This means there shouldn't be any locked profit left.

Impact

Gas.

Recommendation

Remove the call to `lockedProfit()`

Developer Response

Acknowledged.

Informational Findings

1. Informational - Misleading function name and comments

Technical Details

`_onlyOperatorOrController()` has a misleading name:

```
function _onlyOperatorOrController(address _controller) private view {
    BeefySonicStorage storage $ = getBeefySonicStorage();
    if (!$.isOperator[_controller][msg.sender] && _controller != msg.sender)
revert NotAuthorized();
}
```

The function is generic and:

- Takes any address as parameter
- Checks if `msg.sender` is either that address or its authorized operator
- Is used with different types of addresses (owners, controllers)

For example:

```
_onlyOperatorOrController(_owner);  // Checks if msg.sender is owner or owner's
operator
_onlyOperatorOrController(_controller);  // Checks if msg.sender is controller or
controller's operator
```

In `_requestRedeem()` there is a misleading comment:

```
// Ensure the owner is the caller or an authorized operator
_onlyOperatorOrController(_owner);
```

In `_claim()` there is a misleading comment:

```
// we claimed remaining rewards and now set it to claim to false
(bool isOk,) = _validatorStatus(validator.id);
if (!isOk) _setValidatorStatus(i, false, false);
```

Impact

Informational.

Recommendation

Rename the function to reflect its generic nature better:

```
/// @notice Check if the caller is the _account or an authorized operator of the
_account
/// @param _account_ Address to check
function _onlySelfOrOperator(address _account) private view {
    BeefySonicStorage storage $ = getBeefySonicStorage();
    if (_account != msg.sender && !$.isOperator[_account][msg.sender]) revert
NotAuthorized();
}
```

Remove the comment at L230.

Change the comment at L713:

```
- // we claimed remaining rewards and now set it to claim to false
+ // we claimed remaining rewards for inactive validator and now set shouldClaim
to false
```

Developer Response

Fixed in PR#26.

2. Informational - Missing ERC-20 allowance path for `requestRedeem()`

Technical Details

According to ERC-7540 specification:

> *"Redeem Request approval of shares for a msg.sender NOT equal to owner may come either from ERC-20 approval over the shares of owner or if the owner has approved the msg.sender as an operator."*

The current implementation in `_requestRedeem()` only supports operator approval. The contract does not implement the ERC-20 allowance path for share approval, limiting the request redeem functionality to only operator-approved addresses.

Impact

Informational.

Recommendation

Consider adding ERC-20 approval path as specified in ERC-7540.

Developer Response

Fixed in PR#27.

3. Informational - Incorrect implementation of ERC-7540 deposit/mint

Technical Details

The contract implements overloaded `deposit()` and `mint()` method with a `controller` parameter.

However, according to ERC-7540, the `controller` parameter is specifically meant to "discriminate the Request for which the assets should be claimed" in asynchronous deposit flows. Since this contract implements synchronous deposits, the `controller` parameter serves no purpose and could lead to confusion.

Impact

Informational.

Recommendation

Since the contract uses synchronous deposits, you can remove the overloaded methods and use the standard ERC-4626 deposit/mint.

Developer Response

Fixed in PR#28.

4. Informational - Missing ERC-20 interface support

Technical Details

`supportsInterface()` is missing support for the ERC-20 interface:

```
function supportsInterface(bytes4 interfaceId) external pure returns (bool
supported) {
    if (
        interfaceId == 0xe3bc4e65 || // ERC-7540 operator methods
        interfaceId == 0x620ee8e4 || // ERC-7540 async redemption
        interfaceId == 0x2f0a18c5 || // ERC-7575
        interfaceId == 0x01ffc9a7    // ERC-165
```

```
    ) return true;
    return false;
}
```

The contract implements ERC-20 through `ERC20Upgradeable` and `ERC20PermitUpgradeable` but doesn't declare support for its interface ID (`0x36372b07`).

Impact

Informational.

Recommendation

Add ERC-20 interface support:

```
function supportsInterface(bytes4 interfaceId) external pure returns (bool) {
    return
        interfaceId == 0xe3bc4e65 || // ERC-7540 operator methods
        interfaceId == 0x620ee8e4 || // ERC-7540 async redemption
        interfaceId == 0x2f0a18c5 || // ERC-7575
        interfaceId == 0x01ffc9a7 || // ERC-165
        interfaceId == 0x36372b07;   // ERC-20
}
```

Developer Response

Fixed in PR#29.

5. Informational - Remove redundant inheritance

Technical Details

The `BeefySonic` contract inherits from `ERC20Upgradeable` and `ERC20PermitUpgradeable` .

However, `ERC20PermitUpgradeable` already extends `ERC20Upgradeable` , making its explicit inheritance redundant.

Impact

Informational.

Recommendation

Remove the redundant inheritance.

Developer Response

Fixed in PR#30.

6. Informational - Use constants instead of magic numbers

Technical Details

The contract uses several magic numbers without declaring them as named constants:

- L72: `0.1e18` for maximum liquidity fee
- L81: `1 days` for lock duration
- L82: `1e6` for minimum harvest
- L417: `1e18` for precision
- L445: `1e18` for precision
- L653: `1e18` for precision
- L725: `1e18` for precision
- L727: `1e18` for precision
- L805: `1e18` for one share
- L811: `1e18` for one share

Impact

Informational. Magic numbers reduce code readability and maintainability.

Recommendation

Define constants for all magic numbers:

```
uint256 private constant MAX_LIQUIDITY_FEE = 0.1e18;  // 10%
uint256 private constant INITIAL_LOCK_DURATION = 1 days;
uint256 private constant INITIAL_MIN_HARVEST = 1e6;
uint256 private constant PRECISION = 1e18;
uint256 private constant ONE_SHARE = 1e18;
```

Developer Response

Acknowledged.

7. Informational - Incorrect events emission

Technical Details

Several setter functions emit events with incorrect parameter ordering because they use the updated storage value instead of storing the old value before the update:

- `setKeeper()`

```
$.keeper = _keeper;
emit KeeperSet($.keeper, _keeper);  // same value
```

- `setLockDuration()`

```
$.lockDuration = _lockDuration;
emit LockDurationSet($.lockDuration, _lockDuration);  // same value
```

- `setMinHarvest()`

```
$.minHarvest = _minHarvest;
emit MinHarvestSet($.minHarvest, _minHarvest);  // same value
```

Impact

Informational.

Recommendation

Store the old value before updating storage:

```
function setKeeper(address _keeper) external onlyOwner {
    _NoZeroAddress(_keeper);
    BeefySonicStorage storage $ = getBeefySonicStorage();
    address oldKeeper = $.keeper;
    $.keeper = _keeper;
    emit KeeperSet(oldKeeper, _keeper);
}
```

Apply the same pattern to all setter functions that emit events with old and new values.

Developer Response

Fixed in PR#31.

8. Informational - Useless `recoverableAmount`

Technical Details

The struct `Validator` has a `recoverableAmount` attribute that is set inside the function `checkForSlashedValidatorsAndUndelegate()` however it is never read as the amount is recalculated then set to 0 inside the following function `completeSlashedValidatorWithdraw()`.

Additionally, the value of the recoverable amount may change over time; this is because it depends on the `slashingRefundRatio` value set inside the SFC contract. The Sonic multisig sets this value for each validator and is zero by default.

This means that the recoverable amount may change between the first call to `checkForSlashedValidatorsAndUndelegate()` and then the call to `completeSlashedValidatorWithdraw()`. For that reason, it is safer not to store this value in storage and calculate it whenever needed to ensure that you have the latest `slashingRefundRatio` value.

Impact

Informational.

Recommendation

Remove the attribute from the `Validator` struct.

Developer Response

Fixed in PR#34.

9. Informational - Revert if `completeSlashedValidatorWithdraw()` was already called

Technical Details

Currently if `checkForSlashedValidatorsAndUndelegate()` was called for a validator the following function `completeSlashedValidatorWithdraw()` can be called more than once on that same validator.

While the function is `onlyOwner`, a mistake by the beefy admins could reduce `storedTotal` more than it should have and break the invariant, locking some of the user's deposits forever.

Impact

Informational.

Recommendation

Check if `slashedDelegations` is set at the beginning and revert otherwise. At the end of the function, reset `slashedDelegations` to 0.

Developer Response

Fixed in PR#35.

10. Informational - `withdrawDuration()` may change while requests are ongoing

Technical Details

The function `_requestRedeem()` saves the timestamp at which the request can be executed using the `withdrawDuration()` inside `claimableTimestamp`.

However, this value may change if the Sonic admins update it, resulting in an incorrect `claimableTimestamp`.

Users who requested before the `withdrawDuration()` update may think they can claim, but it will revert, or may think they can't claim even though it's actually available, depending on whether the value was increased or decreased.

Impact

Informational.

Recommendation

Consider saving the timestamp at which the request was made and use `withdrawDuration()` inside `_processWithdraw()` instead.

Additionally, consider updating `pendingRedeemRequest()` and `claimableRedeemRequest()`.

Developer Response

Fixed in PR#36.

11. Informational - Update `storedTotal` before undelegating

Technical Details

The function `_requestRedeem()` calls `_getValidatorsToWithdraw()` to get a list of validators it can withdraw from. It then loops through them, calling `undelegate()` and decreasing `storedTotal` and `validator.delegations` for each.

However, before that loop, the function first burns all the shares that are being requested to be redeemed.
This means that if there was any way to reenter the loop, an attacker could exploit the contract by taking advantage of an inflated share value.

While it seems impossible to reenter that loop currently, some contracts are known to implement hooks or callbacks. In this case, the SFC contract has a `stakeSubscriber` that is called during delegations with no enforcement on gas or actions. Currently, that contract is set to `address(0),` but it might change in the future.

Assuming the SFC contract and its callbacks are safe, it is still strongly encouraged to use Checks-Effects-Interactions patterns such as updating the `storedTotal` with the `assets` value calculated at the beginning of the contract before the loop.

Impact

Informational.

Recommendation

Decrease `storedTotal` at the beginning of the function outside the loop.

Developer Response

Fixed in PR#37.

Final remarks

The Beefy protocol on Sonic chain implements a liquid staking solution for Sonic (`S`) tokens, enabling users to earn and auto-compound staking rewards while maintaining liquidity through `beS` tokens. The protocol implements the `ERC-4626` standard for vault

functionality and `ERC-7540` for asynchronous withdrawals. The audit revealed medium-severity issues centered around reward distribution and slashing mechanisms. Test coverage is solid but lacks fuzzing and invariant testing, particularly for complex scenarios involving multiple validators and slashing events. The code structure is generally clean, with good use of access controls and safe math operations. Documentation is needed, especially regarding complex mechanisms like slashing management.