code4rena

# Infrared

## Smart Contract Security Assessment

Version 1.0

Audit dates: Jan 30 — Feb 01, 2025

Audited by: 0xladboy
bytes032

# Contents

# 1. Introduction

## 1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at [https://code4rena.com/zenith](https://code4rena.com/zenith).

## 1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

## 1.3 Risk Classification

| SEVERITY LEVEL | IMPACT: HIGH | IMPACT: MEDIUM | IMPACT: LOW |
|---|---|---|---|
| Likelihood: High | Critical | High | Medium |
| Likelihood: Medium | High | Medium | Low |
| Likelihood: Low | Medium | Low | Low |

# 2. Executive Summary

## 2.1 About Infrared

Infrared is focused on building infrastructure around the Proof of Liquidity (PoL) mechanism pioneered by Berachain. The protocol aims to maximize value capture by providing easy-to-use liquid staking solutions for BGT and BERA, node infrastructure, and vaults. Through building solutions around Proof of Liquidity (PoL), Infrared is dedicated to enhancing the user experience and driving the growth of the Berachain ecosystem.

## 2.2 Scope

| | |
|---|---|
| Repository | infrared-dao/infrared-contracts |
| Commit Hash | 498756a631ad8ae055f1cc2d5980c5c5784983fc |

## 2.3 Audit Timeline

| DATE | EVENT |
|---|---|
| Jan 30, 2025 | Audit start |
| Feb 01, 2025 | Audit end |
| Feb 17, 2025 | Report published |

## 2.4 Issues Found

| SEVERITY | COUNT |
|---|---|
| Critical Risk | 0 |
| High Risk | 0 |
| Medium Risk | 2 |
| Low Risk | 0 |
| Informational | 0 |
| Total Issues | 2 |

# 3. Findings Summary

| ID | DESCRIPTION | STATUS |
|---|---|---|
| M-1 | HarvestBase can still revert unexpectedly | Resolved |
| M-2 | Add incentive lacks access control | Acknowledged |

# 4. Findings

## 4.1 Medium Risk

A total of 2 medium risk findings were identified.

### [M-1] HarvestBase can still revert unexpectedly

| Severity: Medium | Status: Resolved |
| --- | --- |

**Target**

- Infrared.sol#harvestBase

**Severity**

- Impact: Medium
- Likelihood: Medium

**Description**

The finding link is #104

The fix PR is #513

The find demonstrates that the function `harvestBase` is tightly coupled with the function that adds, removes, or replaces the validator.

If the harvestBase reverts, the add / remove / replace function reverts as well.

The revert condition is in the external call `IBerachainBGT(bgt).redeem`,

the redeem function apply modifier `invarantCheck` and `checkUnboostedBalance`

```
    function redeem(
        address receiver,
        uint256 amount
    )
        external
        invariantCheck
        checkUnboostedBalance(msg.sender, amount)
    {
        /// Burn the BGT token from the msg.sender account and reduce the
total supply.
        super._burn(msg.sender, amount);
        /// Transfer the Native token to the receiver.
```

```
        SafeTransferLib.safeTransferETH(receiver, amount);
        emit Redeem(msg.sender, receiver, amount);
    }
```

Unboosted Balance Check: The unboosted balance of Infrared must be greater than or equal to the amount being redeemed.

BGT.sol#L449

```
function _checkUnboostedBalance(address sender, uint256 amount) private
view {
    if (unboostedBalanceOf(sender) < amount)
NotEnoughBalance.selector.revertWith();
}
function unboostedBalanceOf(address account) public view returns
(uint256) {
    UserBoost storage userBoost = userBoosts[account];
    (uint128 boost, uint128 _queuedBoost) = (userBoost.boost,
userBoost.queuedBoost);
    return balanceOf(account) - boost - _queuedBoost;
}
```

Invariant Check: The BGT contract must hold a BERA balance greater than or equal to its total supply.

BGT.sol#L461

```
function _invariantCheck() private view {
    if (address(this).balance < totalSupply())
InvariantCheckFailed.selector.revertWith();
}
```

So we need to check these two invariants before calling redeem to ensure redeem not revert.

This is the fix code

```
    function harvestBase(address ibgt, address bgt, address ibera)
        external
        returns (uint256 bgtAmt)
    {
        // Since BGT balance has accrued to this contract, we check
  for what we've already accounted for
        uint256 minted = IInfraredBGT(ibgt).totalSupply();
```

```
        // The balance of BGT in the contract is the rewards
accumilated from base rewards since the last harvest
        // Since is paid out every block our validators propose (have
a `Distributor::distibuteFor()` call)
        uint256 balance = IBerachainBGT(bgt).balanceOf(address(this));
        if (balance == 0) return 0;

        // @dev The amount that will be minted is the difference
between the balance accrued since the last harvestVault and the
current balance in the contract.
        // This difference should keep getting bigger as the contract
accumulates more bgt from `Distributor::distibuteFor()` calls.
        bgtAmt = balance - minted;

 +       // @dev ensure that the `BGT::redeem` call won't revert if
there is no BERA backing it.
 +       // This is not unlikely since
https://github.com/berachain/beacon-kit slots/blocks are not
consistent there are times
 +       // where the BGT rewards are not backed by BERA, in this case
the BGT rewards are not redeemable.
 +


 +       if (bgtAmt > bgt.balance) return 0;




        IBerachainBGT(bgt).redeem(IInfraredBERA(ibera).receivor(),
bgtAmt);

    }


```

The newly added check still does not fully re-implement the two modifier function checks in the redeem function, so redeem can still revert.

**Recommendation:**

Fix:

```
    // comply with the modifier checkUnboostedBalance
```

```
    // if unboostedBalanceOf(sender) < amount, revert
    if IBerachainBGT(bgt).unboostedBalanceOf(address(this)) < bgtAmt)
return 0;

    // comply with modifier invariantCheck
    // if address(this).balance < totalSupply(), revert, address(this) is
bgt contract.

    if(address(bgt).balance < bgt.totalSupply()) return 0;

     try IBerachainBGT(bgt).redeem(IInfraredBERA(ibera).receivor(),
bgtAmt) {} catch {} // catch try can be used for additional security.
```

**Infra:** Fixed with the following PR-549 using try-catch

**Zenith:** Verified.

## [M-2] Add incentive lacks access control

| Severity: Medium | Status: Acknowledged |
| --- | --- |

**Target**

- Infrared.sol#addIncentives

**Severity:**

- Impact: Medium
- Likelihood: Medium

**Description:**

The primary finding is Finding #445

The fix PRs are #427 and #423

The code ensures the added reward cannot decrease the reward rate. However, the reward period is still extended.

A lot of duplicates recommend the same fix, which is not implemented.

Access control is still missing in the function addIncentives

**Recommendation:**

Recommendations from findings 4 and 11 should be followed. Both findings offer the same recommendation.

- Finding #4
- Finding #11

Add Access Control: Apply appropriate access control (e.g., onlyGovernor) to the Infrared::addIncentives() function

Enforce Minimum Token Amount:Set a reasonable minimum token amount for InfraredVault::notifyRewardAmount() to ensure calls with insignificant amounts (e.g., 1 wei) are rejected.

MultiRewards has been forked from Curve and Curve implements access controls for the notifyRewardAmount() function.

https://cantina.xyz/code/ac5f64e6-3bf2-4269-bbb0-4bcd70425a1d/findings/650

synthetix/StakingRewards and curvefi/multi-rewards allows only owners to notify rewards, to counter issues like this.

It is also highly recommend to add testing to the new PR fix.

**Infra:** Acknowledged - the fix will be implemented in a future release.

**Zenith:** The code in branch access-controll-addIncentives implements the fix. The fix commit is 099ce797a9ea189cb87704ed723bd75e6a48c33d

The main branch addIncentive `addIncentive` function still has no access control.