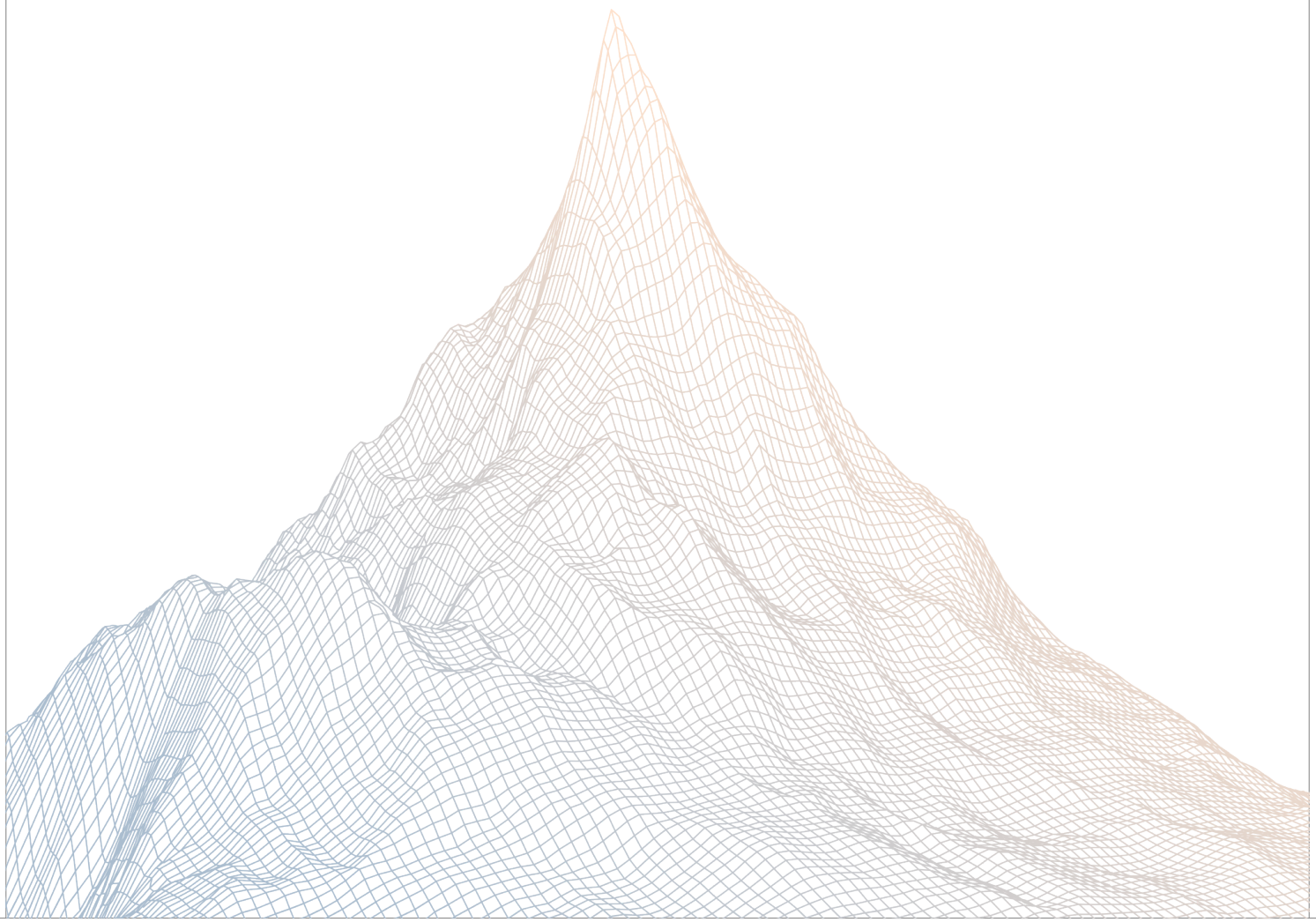
Zenith

Infrared

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: August 20th to August 20th, 2025

AUDITED BY: Matte
ether_sky

Contents

1 Introduction

1.1 About Zenith

Zenith assembles auditors with proven track records: finding critical vulnerabilities in public audit competitions.

Our audits are carried out by a curated team of the industry's top-performing security researchers, selected for your specific codebase, security needs, and budget.

Learn more about us at https://zenith.security.

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

The engagement involved a review of the following targets:

Target	infrared-contracts
Repository	https://github.com/infrared-dao/infrared-contracts
Commit Hash	`aec31881f8cf9177242502ce215004adbb58b0d3`
Files	`Changes in PR 628`

2.3 Audit Timeline

August 20, 2025	Audit start
August 20, 2025	Audit end
August 28, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	0
Informational	0
Total Issues	**1**

3

Findings Summary

ID	Description	Status
M-1	In some cases, redemption may be reverted in the Redeemer	Acknowledged

4

Findings

4.1 Medium Risk

A total of 1 medium risk findings were identified.

[M-1] In some cases, redemption may be reverted in the Redeemer

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- Redeemer.sol

Description:

In the `redeemIbgtForBera` function, the `redeemed amount` should be available on `Infrared` as `unboosted BGT`, as noted in the comments.

- Redeemer.sol#L53

```
function redeemIbgtForBera(uint256 amount) external {
    if (amount == 0) revert InvalidAmount();
    // amount to be redeemed must be available on infrared as unboosted BGT
    uint256 unboostedBalance = IInfraredBGT(ibgt).totalSupply()
        - (
            IBerachainBGT(bgt).boosts(address(infrared))
                + IBerachainBGT(bgt).queuedBoost(address(infrared))
        );
    if (unboostedBalance < amount) revert InsufficientUnboostedBGT();
    ...
}
```

The `unboosted BGT` amount is calculated as follows in the `BGT`.

- BGT.sol#L431

```
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;
}
```

The `total supply` of the `iBGT token` is usually less than `Infrared`'s `BGT balance`. These values are synchronized through functions like `auctionBase`.

- RewardsLib.sol#L349

```
function auctionBase(
    address ibgt,
    address bgt,
    address harvestBaseCollector
) 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 accumulated 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 accumilates
    more bgt from `Distributor::distibuteFor()` calls.
    bgtAmt = balance - minted;

    // @dev mint equivilent of iBGT to the harvestBaseCollector
    IInfraredBGT(ibgt).mint(harvestBaseCollector, bgtAmt);
}
```

As a result, `redemption` can still be reverted even if there are enough `unboosted BGTs` available.

The `Redeemer` is one of `Infrared`'s `keepers`, and any `keeper` can call `redeemIbgtForBera`.

- InfraredV1_8.sol#L733

```
function redeemIbgtForBera(address bgt, address ibgt, uint256 amount)
    external
```

```
{
    ERC20(ibgt).safeTransferFrom(msg.sender, address(this), amount);
    IInfraredBGT(ibgt).burn(amount);
    IBerachainBGT(bgt).redeem(msg.sender, amount);
}
```

So, even though the intention is to sync with the `total iBGT supply`, other `keepers` may not consider `iBGT`'s `total supply` at all.

Recommendations:

```
function redeemIbgtForBera(uint256 amount) external {
    if (amount == 0) revert InvalidAmount();
    // amount to be redeemed must be available on infrared as unboosted BGT
    uint256 unboostedBalance = IInfraredBGT(ibgt).totalSupply()

    uint256 unboostedBalance = IBerachainBGT(bgt).balanceOf(address(infrared))
        - (
            IBerachainBGT(bgt).boosts(address(infrared))
                + IBerachainBGT(bgt).queuedBoost(address(infrared))
        );
    if (unboostedBalance < amount) revert InsufficientUnboostedBGT();

}
```

Infrared: Acknowledged.

Finding accurate but we are not going to change it because:

- redemptions will be kept private (onlyKeeper for time being), where we manage how much we want to redeem.
- 100k BGT buffer is applied at operational level.
- we do not want to take from base rewards.