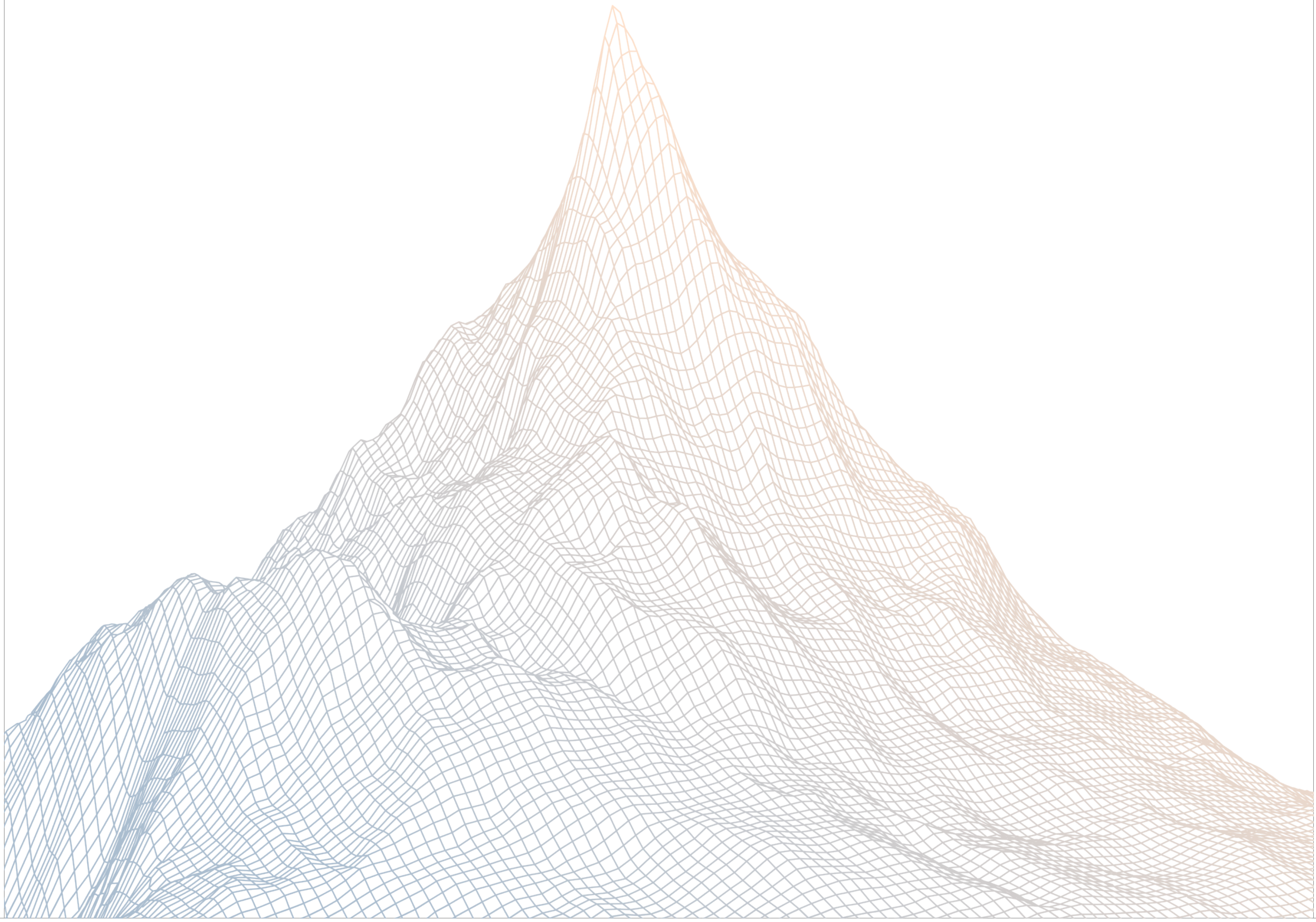
Zenith

Infrared

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: September 24th to September 24th, 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	`3295ec7e5ee71c5127bbc58e6b27de1beb63ca72`
Files	`src/periphery/MerkleDistributor.sol Diff from b6bdd3dc0a295dd21fe3993edb759131c942a76d`

2.3 Audit Timeline

September 24, 2025	Audit start
September 24, 2025	Audit end
September 30, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	1
Informational	4
Total Issues	**5**

3

Findings Summary

ID	Description	Status
L-1	The setTotalAllocated function should include a check to validate the token balance	Resolved
I-1	There are unused events that can be removed	Resolved
I-2	The constructor does not validate the window parameter	Resolved
I-3	Remove the unnecessary condition check in the canClaim function	Resolved
I-4	The canClaim function must return false if the distributor is in a paused state	Resolved

4 Findings

4.1 Low Risk

A total of 1 low risk findings were identified.

[L-1] The setTotalAllocated function should include a check to validate the token balance

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- MerkleDistributor.sol

Description:

The `owner` has the ability to change `totalAllocated`.

- MerkleDistributor.sol#L239

```
function setTotalAllocated(uint256 _totalAllocated) external onlyOwner {
    totalAllocated = _totalAllocated;
}
```

Also the `owner` can recover any tokens. If the recovered token is the `distributor`'s main token, the `distributor` must still retain enough tokens to cover `allocations`.

- MerkleDistributor.sol#L205

```
function recoverERC20(address tokenAddress, address to)
    external
    onlyOwner
{
    ERC20 recoveryToken = ERC20(tokenAddress);
    uint256 balance = recoveryToken.balanceOf(address(this));

    if (balance == 0) revert ZeroAmount();

    // Prevent recovering distribution tokens before deadline unless it's
    excess
    if (tokenAddress == address(token) && block.timestamp <= claimDeadline)
    {
        // Only allow recovering excess tokens (tokens beyond what's
    allocated)
        uint256 requiredBalance = totalAllocated - totalClaimed;
        if (balance <= requiredBalance) {
            revert DeadlineNotReached();
        }
        // Recover only the excess
        balance = balance - requiredBalance;
    }

    recoveryToken.safeTransfer(to, balance);
    emit EmergencyWithdraw(tokenAddress, to, balance);
}
```

If excess tokens are recovered and `totalAllocated` is later increased, there may not be enough tokens left to `distribute`.

Recommendations:

```
function setTotalAllocated(uint256 _totalAllocated) external onlyOwner {
    totalAllocated = _totalAllocated;

    if (totalAllocated < totalClaimed) revert();
    uint256 balance = token.balanceOf(address(this));
    if (balance < totalAllocated - totalClaimed) revert();
}
```

Infrared: Resolvedd with PR-636.

Zenith: Verified.

4.2 Informational

A total of 4 informational findings were identified.

[I-1] There are unused events that can be removed

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- MerkleDistributor.sol

Description:

The following `event` and `error` are unused.

- MerkleDistributor.sol#L47

```
event DeadlineExtended(uint256 newDeadline);
error TransferFailed();
```

Recommendations:

```
event DeadlineExtended(uint256 newDeadline);
error TransferFailed();
```

Infrared: Resolved with PR-636.

Zenith: Verified.

[I-2] The constructor does not validate the window parameter

`SEVERITY`: Informational	`IMPACT`: Informational
`STATUS`: Resolved	`LIKELIHOOD`: Low

Target

- MerkleDistributor.sol

Description:

In the `constructor`, all `input parameters` are validated except for `window`.

- MerkleDistributor.sol#L87

```
constructor(
    address _token,
    bytes32 _merkleRoot,
    address _owner,
    uint256 window,
    uint256 _totalAllocated
) Owned(_owner) {
    if (_token == address(0)) revert ZeroAddress();
    if (_merkleRoot == bytes32(0)) revert InvalidMerkleRoot();
    if (_owner == address(0)) revert ZeroAddress();
    if (_totalAllocated == 0) revert ZeroAmount();

    token = ERC20(_token);
    merkleRoot = _merkleRoot;
    claimDeadline = block.timestamp + window;
    totalAllocated = _totalAllocated;
}
```

Recommendations:

A `minimum value check` (or similar `validation`) should also be added for `window`.

Infrared: Resolved with PR-636.

Zenith: Verified.

[I-3] Remove the unnecessary condition check in the canClaim function

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- MerkleDistributor.sol

Description:

In the `canClaim` function, the `amount` is always greater than `claimed[account]` because of the check at `line 149`. As a result, the difference between them at `line 160` will always be greater than 0.

- MerkleDistributor.sol#L149

```
function canClaim(
    address account,
    uint256 amount,
    bytes32[] calldata merkleProof
) external view returns (bool _canClaim, uint256 claimableAmount) {
    if (block.timestamp > claimDeadline) {
        return (false, 0);
    }

149: if (claimed[account] >= amount) {
        return (false, 0);
    }

    bytes32 leaf = keccak256(abi.encode(account, amount));
    bool validProof = MerkleProofLib.verify(merkleProof, merkleRoot, leaf);

    if (!validProof) {
        return (false, 0);
    }

160: claimableAmount = amount - claimed[account];
    _canClaim = claimableAmount > 0 ? true : false;
}
```

Recommendations:

```
function canClaim(
    address account,
    uint256 amount,
    bytes32[] calldata merkleProof
) external view returns (bool _canClaim, uint256 claimableAmount) {
    if (block.timestamp > claimDeadline) {
        return (false, 0);
    }

    if (claimed[account] >= amount) {
        return (false, 0);
    }

    bytes32 leaf = keccak256(abi.encode(account, amount));
    bool validProof = MerkleProofLib.verify(merkleProof, merkleRoot, leaf);

    if (!validProof) {
        return (false, 0);
    }

    claimableAmount = amount - claimed[account];
    _canClaim = claimableAmount > 0 ? true : false;
    _canClaim = true;
}
```

Infrared: Resolved with PR-636.

Zenith: Verified.

[I-4] The canClaim function must return false if the distributor is in a paused state

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: Low

Target

- MerkleDistributor.sol

Description:

`Claims` are not allowed when the `distributor` is `paused`.

- MerkleDistributor.sol#L113

```
function claim(uint256 amount, bytes32[] calldata merkleProof)
    external
    nonReentrant
    whenNotPaused
{
    _claim(msg.sender, amount, merkleProof);
}
```

Since users check their `claimable status` through the `canClaim` function, this function should also return `false` whenever the `distributor` is `paused`.

- MerkleDistributor.sol#L140-L144

```
function canClaim(
    address account,
    uint256 amount,
    bytes32[] calldata merkleProof
) external view returns (bool _canClaim, uint256 claimableAmount) {
    if (block.timestamp > claimDeadline) {
        return (false, 0);
    }
    ...
}
```

Recommendations:

```
function canClaim(
    address account,
    uint256 amount,
    bytes32[] calldata merkleProof
) external view returns (bool _canClaim, uint256 claimableAmount) {
    if (paused()) {
        return (false, 0);
    }

    if (block.timestamp > claimDeadline) {
        return (false, 0);
    }
    ...
}
```

Infrared: Resolved with PR-636.

Zenith: Verified.