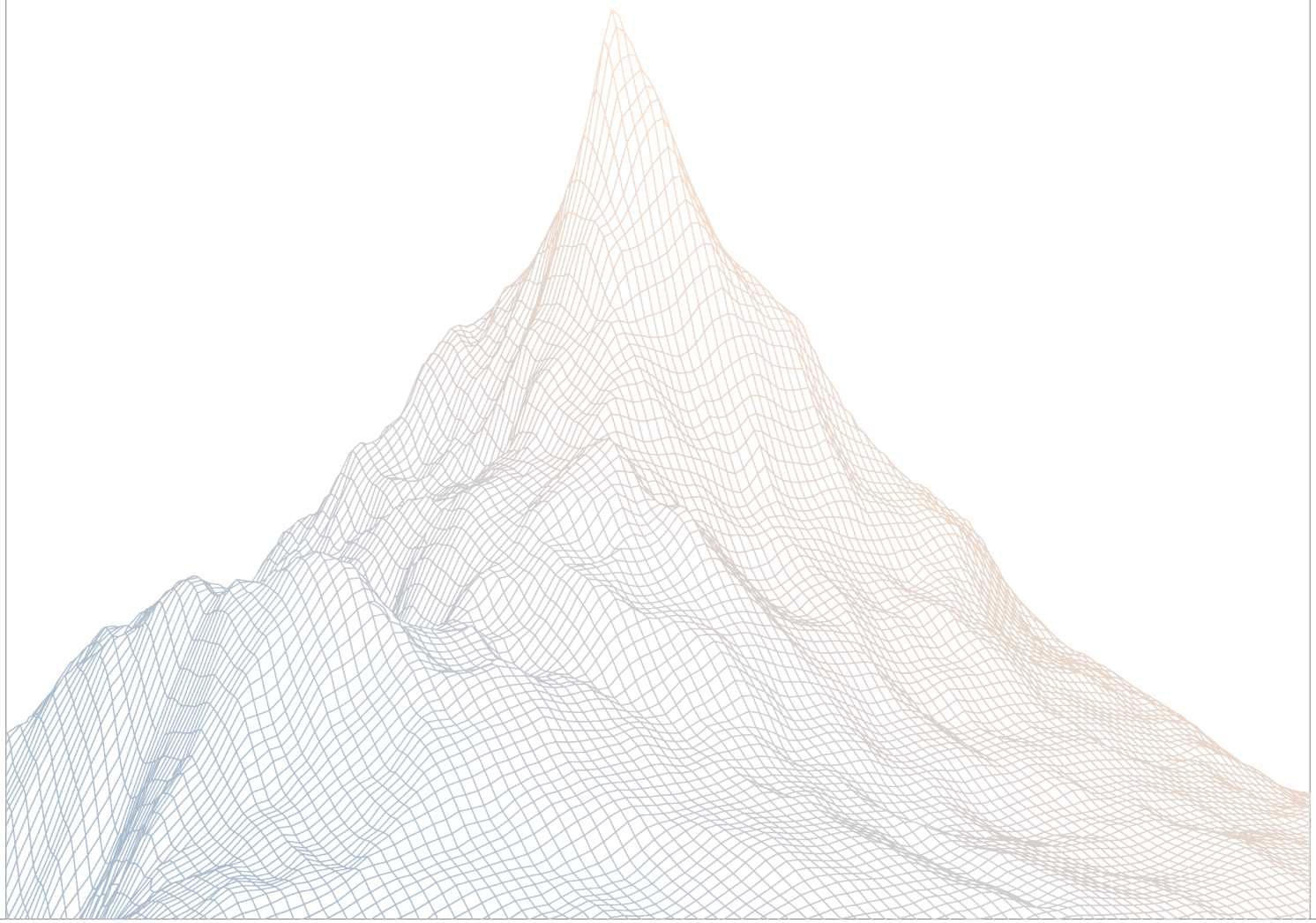
Zenith

Spectral

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: May 28th to May 29th, 2025

AUDITED BY: ether_sky
sorryNotsorry

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 Spectral

At Spectral, we're pioneering the Onchain Agent Economy—a revolutionary paradigm where anyone can build agentic companies composed of multiple autonomous AI agents.

Imagine a decentralized future where intelligent agents not only navigate the crypto landscape 24/7 but also collaborate towards a common goal, handling workflows like hiring, firing, performance management, deposits, and rewards distribution and providing real world utility to Web3 users.

Our mission is to advance and simplify onchain AI, breaking down technical barriers so that whether you're a normie or a degen, risk off or risk on, you can tap into the full power of onchain AI agents.

2.2 Scope

The engagement involved a review of the following targets:

Target	autonomous-agent-contracts
Repository	https://github.com/Spectral-Finance/autonomous-agent-contracts
Commit Hash	`a62aa05d2f327b226d8031cbcf28b51e6f1d387f`
Files	`Changes in PR-29`

2.3 Audit Timeline

May 28th, 2025	Audit start
May 29th, 2025	Audit end
May 30th, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	1
Medium Risk	0
Low Risk	1
Informational	2
Total Issues	**4**

3

Findings Summary

ID	Description	Status
H-1	The withdrawn amount should be tracked separately for each token	Resolved
L-1	The rewards could be withdrawn for both tokens if the same Merkle Root is updated	Resolved
I-1	MerkleRootUpdated event omits the token address	Resolved
I-2	withdrawPoints doesn't implement nonReentrant modifier	Resolved

4

Findings

4.1 High Risk

A total of 1 high risk findings were identified.

[H-1] The withdrawn amount should be tracked separately for each token

SEVERITY: High	IMPACT: High
STATUS: Resolved	LIKELIHOOD: Medium

Target

- OctoDistributor.sol

Description:

There are two tokens, `A` and `B`. A user **U** is granted points to withdraw **100 A** and **200 B**. First, the user calls the `withdrawPoints` function to withdraw **200 B**. As a result, `userPointsWithdrawn[U]` becomes **200**. Now, if the user tries to withdraw **100 A**, the transaction will revert — even though they have enough points for `A`.

- OctoDistributor.sol#L524-L531

```
function withdrawPoints(
    address _token,
    uint256 amount,
    uint256 totalAmount,
    bytes32[] calldata merkleProof
) external {
    // Verify the user has this allocation in the merkle tree
    bytes32 leaf = keccak256(abi.encodePacked(msg.sender, totalAmount));
    require(
        MerkleProofUpgradeable.verify(merkleProof,
    pointsMerkleRoots[_token], leaf),
        "Invalid merkle proof"
    );

    // Check how much is already withdrawn and how much can be withdrawn
    uint256 alreadyWithdrawn = userPointsWithdrawn[msg.sender];
    require(
```

```
@->         alreadyWithdrawn + amount <= totalAmount,
            "Attempting to withdraw more than allocated"
        );

        // Update the withdrawn amount
@->     userPointsWithdrawn[msg.sender] = alreadyWithdrawn + amount;

        // Transfer the tokens
        IERC20Upgradeable(_token).safeTransfer(msg.sender, amount);

        emit RewardsClaimed(msg.sender, amount);
    }
```

This happens because `userPointsWithdrawn` is being tracked as a single value, without distinguishing between tokens. To fix this, *`userPointsWithdrawn`* **should be tracked separately for each token.**

Recommendations:

```
- mapping(address => uint256) public userPointsWithdrawn;
+ mapping(address => address => uint256) public userPointsWithdrawn;

function withdrawPoints(
    address _token,
    uint256 amount,
    uint256 totalAmount,
    bytes32[] calldata merkleProof
) external {
    // Verify the user has this allocation in the merkle tree
    bytes32 leaf = keccak256(abi.encodePacked(msg.sender, totalAmount));
    require(
        MerkleProofUpgradeable.verify(merkleProof,
    pointsMerkleRoots[_token], leaf),
        "Invalid merkle proof"
    );

    // Check how much is already withdrawn and how much can be withdrawn
-    uint256 alreadyWithdrawn = userPointsWithdrawn[msg.sender];
+    uint256 alreadyWithdrawn = userPointsWithdrawn[msg.sender][_token];

    require(
        alreadyWithdrawn + amount <= totalAmount,
        "Attempting to withdraw more than allocated"
    );
```

```
    // Update the withdrawn amount
-    userPointsWithdrawn[msg.sender] = alreadyWithdrawn + amount;
+    userPointsWithdrawn[msg.sender][_token] = alreadyWithdrawn + amount;

    // Transfer the tokens
    IERC20Upgradeable(_token).safeTransfer(msg.sender, amount);

    emit RewardsClaimed(msg.sender, amount);
}
```

Spectral: Resolved with @9fce8cd3692...

Zenith: Verified.

4.2 Low Risk

A total of 1 low risk findings were identified.

[L-1] The rewards could be withdrawn for both tokens if the same Merkle Root is updated

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

Target

- OctoDistributor.sol

Description:

As per the information, the Merkle Roots will be updated weekly. If the admin (accidentally) uploads the same Merkle root hash for two different tokens (e.g. Spectral and USDC) every address appearing in that root can withdraw the same allocation once per token.

Recommendations:

Include the `_token` in the leaf;

```
bytes32 leaf = keccak256(abi.encodePacked(msg.sender, totalAmount));

    bytes32 leaf = keccak256(abi.encodePacked(msg.sender, _token, totalAmount));
```

Spectral: Resolved with @89e7b1c07....

Zenith: Verified.

4.3 Informational

A total of 2 informational findings were identified.

[I-1] MerkleRootUpdated event omits the token address

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: High

Target

- OctoDistributor.sol
- OctoDistributor.sol

Description:

The `MerkleRootUpdated` event only logs the old and the new root hashes:

```
event MerkleRootUpdated(bytes32 oldRoot, bytes32 newRoot);
```

Without the `_token` field, off-chain monitors can't tell which reward pool was touched.

Recommendations:

Include the `_token` in the event;

```
event MerkleRootUpdated(bytes32 oldRoot, bytes32 newRoot);
event MerkleRootUpdated(address indexed token, bytes32 oldRoot, bytes32
    newRoot);
```

Spectral: Resolved with @05412ee72....

Zenith: Verified.

[I-2] withdrawPoints doesn't implement nonReentrant modifier

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

Target

- OctoDistributor.sol

Description:

While the other functions, `withdraw`, `withdrawAllAgentTokens`, `withdrawAllAgentTokensByAddresses`, `transferTradingRewards`, `transferHiringDistributions` have `nonReentrant` modifier, `withdrawPoints` doesn't implement it.

Recommendations:

Recommended to add the modifier to `withdrawPoints` too for the whole coverage.

Spectral: Resolved with @ef51279de....

Zenith: Verified.