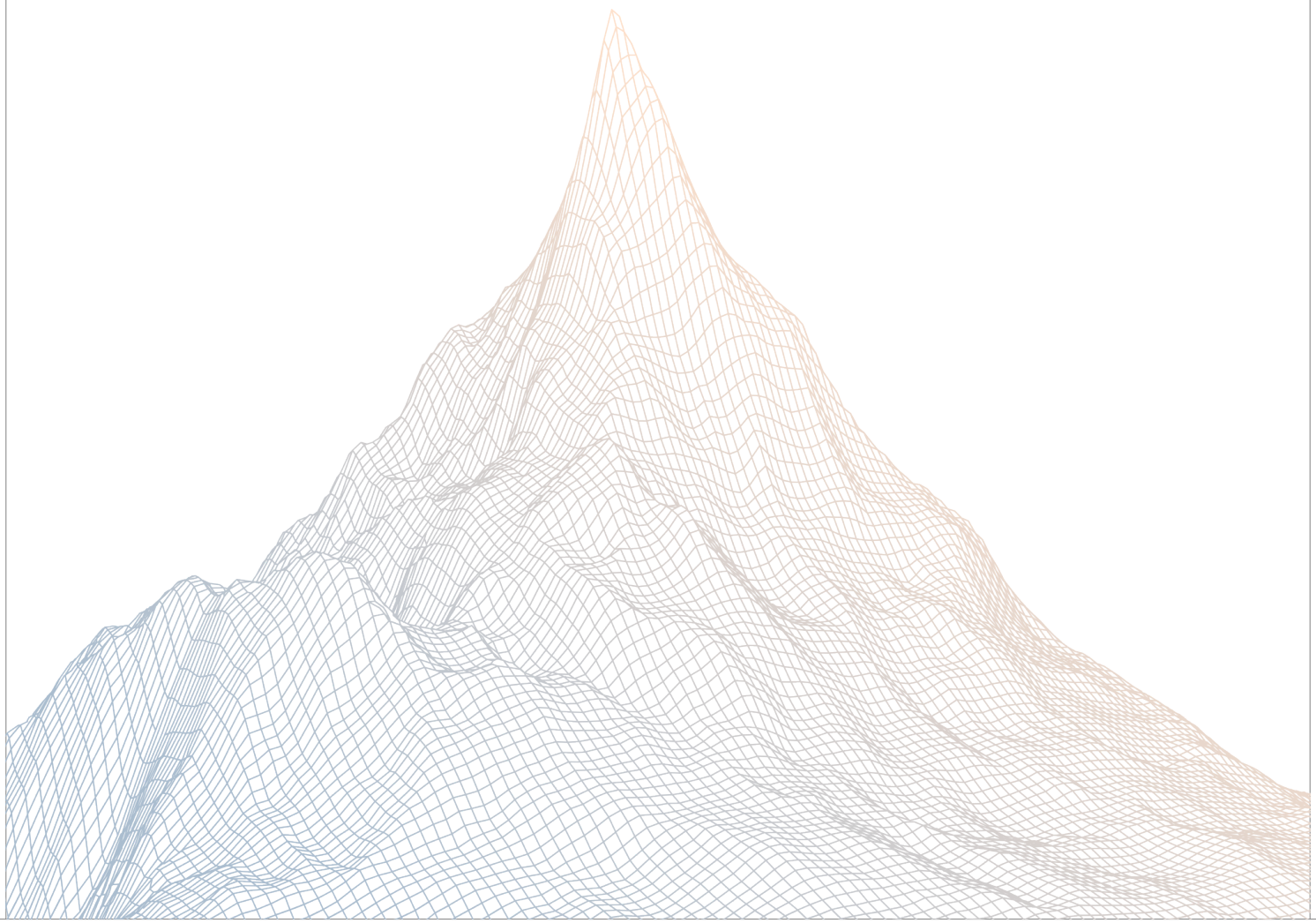
Zenith

Spectral

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: April 23rd to April 24th, 2025

AUDITED BY: ether_sky
sorryNotSorry

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.

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	`6a9e12846746626365289106664fbde2646541f4`
Files	`Changes in PR #26`

2.3 Audit Timeline

April 23rd, 2025	Audit start
April 24th, 2025	Audit end
April 28th, 2025	Report published

2.4 Issues Found

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

3

Findings Summary

ID	Description	Status
H-1	Excessive Fixed Slippage (20%) in Protocol Tax Swaps Creates Persistent MEV Vulnerability	Resolved
H-2	The trading rewards are distributed incorrectly when contributors have zero elements	Resolved
L-1	Hardcoded Uniswap V3 Fee Tier May Not Be Optimal	Acknowledged
I-1	The TRADING_REWARDS_SPECTRA_CUT is not used	Resolved

4

Findings

4.1 High Risk

A total of 2 high risk findings were identified.

[H-1] Excessive Fixed Slippage (20%) in Protocol Tax Swaps Creates Persistent MEV Vulnerability

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

Target

- AutonomousAgentDeployer.sol

Description

The `swapExactTokensForSPEC` function performs two swaps: one for the user's tokens and another for the protocol's tax tokens. While the user's swap uses their specified `amountOutMin` for slippage protection, the protocol's tax swap uses a hardcoded 20% slippage:

```
// First get expected amount
uint256 protocolSpecAmount = getSPECReceived(protocolSwapAmount, fromToken);

// Then execute swap with 20% slippage
uniswapRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(
    protocolSwapAmount,
    (protocolSpecAmount * 80) / 100,  // 20% slippage allowed
    path,
    address(this),
    deadline
);
```

This creates a highly profitable MEV opportunity that will be exploited by sandwich bots on every protocol tax swap:

Bot frontruns the transaction with a large swap to move the price Protocol's tax swap executes with up to 20% slippage Bot backruns to restore price, capturing up to 20% of the protocol's tax tokens

This will result in the protocol consistently losing up to 20% of its tax revenue to MEV bots. The high fixed slippage essentially becomes a "MEV bot fee" paid on every trade.

Recommendations

At minimum, reduce the hardcoded slippage to a reasonable value:

```
// Change from 80% to 98% (2% slippage)
uint256 minOut = (protocolSpecAmount * 98) / 100;
```

Spectral: Resolved with @64948f6435...

Zenith: Verified.

[H-2] The trading rewards are distributed incorrectly when contributors have zero elements

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

Target

- OctoDistributor.sol

Description:

The `transferTradingRewards` function allows the `admin` to distribute `trading rewards` to multiple entities.

- OctoDistributor.sol#L288

```
function transferTradingRewards(
    address agentToken,
    uint256 usdcAmount,
    TradingDistribution[] calldata contributors,
    address altTreasury
) external nonReentrant onlyAdmin() {
    uint256 remaining = usdcAmount;
    if(keccak256(abi.encodePacked("SPECTR")) ==
    keccak256(abi.encodePacked(IAgentToken(agentToken).symbol()))) {
266:        uint256 employeesAmount = (usdcAmount
    * parameters[Parameter.TRADING_REWARDS_EMPLOYEES_CUT]) / PRECISION;
267:        uint256 socialContributorsAmount = (usdcAmount * 2500)
    / PRECISION;
        uint256 stakersAmount = (usdcAmount * 2500) / PRECISION;
        remaining -= stakersAmount;
        ...
        uint256 perEmployeeAmount = employeesAmount / employeeCount;
279:        remaining -= (perEmployeeAmount * employeeCount);
        for (uint256 i = 0; i < employeeCount; i++) {
            address employee =
    _getAnsAddress(IAgenticCompany(systemAddresses.agenticCompanyFactory.getCompanyA
            userBalances[employee].usdc += perEmployeeAmount;
            beneficiaries[i] = employee;
            amounts[i] = perEmployeeAmount;
```

```
        }
        if(contributorCount == 0) {
            require(altTreasury != address(0), "ALT_TREASURY_ZERO_ADDRESS");
288:            usdc_token.safeTransfer(altTreasury,
    socialContributorsAmount);
        }else
        {
            ...
        }
        emit DistributeTradingRewards(agentToken, usdcAmount, beneficiaries,
    amounts);
    } else {
        ...
    }
    require(remaining >= (usdcAmount
    * parameters[Parameter.TRADING_REWARDS_TREASURY_CUT]) / PRECISION,
    "INSUFFICIENT_TREASURY_CUT");
315:    usdc_token.safeTransfer(systemAddresses.spectralTreasury,
    remaining);
}
```

At `line 266`, the `rewards` for `employees` are calculated, and at `line 267`, the `rewards` for `contributors` are determined. The `remaining rewards` represent the 'undistributed portion. These `remaining rewards` are eventually distributed at `line 315`. However, at `line 288`, the `remaining` value is not updated when the `contributors` array is empty. Since this `25%` of the total `rewards` is excluded from the `remaining` calculation, it results in these `rewards` being distributed twice—specifically to `altTreasury` and `spectralTreasury`.

Recommendations:

```
function transferTradingRewards(
    address agentToken,
    uint256 usdcAmount,
    TradingDistribution[] calldata contributors,
    address altTreasury
) external nonReentrant onlyAdmin() {
    if(keccak256(abi.encodePacked("SPECTR")) ==
    keccak256(abi.encodePacked(IAgentToken(agentToken).symbol()))) {
        ...
        if(contributorCount == 0) {
            require(altTreasury != address(0), "ALT_TREASURY_ZERO_ADDRESS");
            usdc_token.safeTransfer(altTreasury, socialContributorsAmount);
+ ^^I^^I^^I remaining -= socialContributorsAmount;
        }else
```

```
        {
            ...
        }
        emit DistributeTradingRewards(agentToken, usdcAmount, beneficiaries,
    amounts);
    } else {
        ...
    }
    require(remaining >= (usdcAmount
    * parameters[Parameter.TRADING_REWARDS_TREASURY_CUT]) / PRECISION,
    "INSUFFICIENT_TREASURY_CUT");
    usdc_token.safeTransfer(systemAddresses.spectralTreasury, remaining);
}
```

Spectral: Resolved with @725af30f64...

Zenith: Verified.

4.2 Low Risk

A total of 1 low risk findings were identified.

[L-1] Hardcoded Uniswap V3 Fee Tier May Not Be Optimal

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

Target

- AutonomousAgentDeployer.sol

Description

The `swapETHForSPEC` function uses a hardcoded fee tier (0.3%) for Uniswap V3 swaps:

```
ISwapRouter.ExactInputSingleParams memory params
    = ISwapRouter.ExactInputSingleParams({
    tokenIn: WETH,
    tokenOut: spectralToken,
    fee: 3000, <<
    recipient: address(this),
    amountIn: ethAmount,
    // ...
});
```

While 0.3% is a common fee tier, it may not always be the most efficient option. Lower fee tiers (0.05% or 0.01%) might offer better rates if SPEC/WETH becomes a stable pair and higher fee tiers (1%) might have more liquidity during high volatility. The Uniswap governance can remove the 0.3% tier too. After all, the most liquid pool for SPEC/WETH might use a different fee tier.

However, since the contract is upgradeable, this falls into a low severity issue.

Recommendations

Despite the upgradeability, it would still be better to:

1. Make the fee tier configurable:

```
uint24 public uniswapFeeTier = 3000;

function setFeeTier(uint24 _feeTier) external onlyOwner {
    require(_feeTier == 500 || _feeTier == 3000 || _feeTier == 10000,
    "Invalid fee tier");
    uniswapFeeTier = _feeTier;
}
```

2. Add fee tier querying:

```
function getBestFeeTier(address tokenIn, address tokenOut)
    internal view returns (uint24) {
    // Query liquidity across fee tiers and return the most liquid one
    // This would require additional interface implementations
}
```

This would provide flexibility without requiring contract upgrades for fee optimizations.

Spectral: Acknowledged, we are liquidity providing on uniswap for now on and we would upgrade when needed.

4.3 Informational

A total of 1 informational findings were identified.

[I-1] The TRADING_REWARDS_SPECTRA_CUT is not used

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

Target

- OctoDistributor.sol

Description:

The `TRADING_REWARDS_SPECTRA_CUT` `percent` is initialized as `0` in the `constructor` but can be adjusted by the `owner` using the `setParameters` function.

- OctoDistributor.sol#L194

```
function setParameters(
    Parameter[] calldata _parameters,
    uint256[] calldata _values
) external onlyOwner {
    // Parameters need to be checked:
    // parameters[Parameter.TRADING_REWARDS_TREASURY_CUT] +
    parameters[Parameter.TRADING_REWARDS_CREATOR_CUT] = 10000;
    // parameters[Parameter.TRADING_REWARDS_TREASURY_CUT] +
    parameters[Parameter.TRADING_REWARDS_SPECTRA_CUT] +
    parameters[Parameter.TRADING_REWARDS_EMPLOYEES_CUT] = 10000;
    require(_parameters.length == _values.length, "INVALID_LENGTH");
    for (uint256 i = 0; i < _parameters.length; i++) {
        require(_values[i] <= PRECISION, "PARAMETER_OUT_OF_RANGE");
        parameters[_parameters[i]] = _values[i];
        emit SetParameter(_parameters[i], _values[i]);
    }
    require(
        parameters[Parameter.TRADING_REWARDS_TREASURY_CUT] +
            parameters[Parameter.TRADING_REWARDS_CREATOR_CUT] ==
            PRECISION,
```

```
            "INVALID_CREATOR_CUT"
        );
        require(
            parameters[Parameter.TRADING_REWARDS_TREASURY_CUT] +
                parameters[Parameter.TRADING_REWARDS_SPECTRA_CUT] +
                parameters[Parameter.TRADING_REWARDS_EMPLOYEES_CUT] ==
                PRECISION / 2,
            "INVALID_SPECTRA_CUT"
        );
    }
```

However, it remains unused in the `transferTradingRewards` function, resulting in the `rewards` for this parameter being redirected to the `treasury`.

Recommendations:

The `parameter` can either be removed entirely or utilized correctly when transferring `trading rewards`. Additionally, please update the comments in the `setParameters` function to reflect that the sum of the three `parameters` should be `5000`, not `10000`, as currently indicated in the code.

Spectral: Resolved with @0baa41333f....

Zenith: Verified.