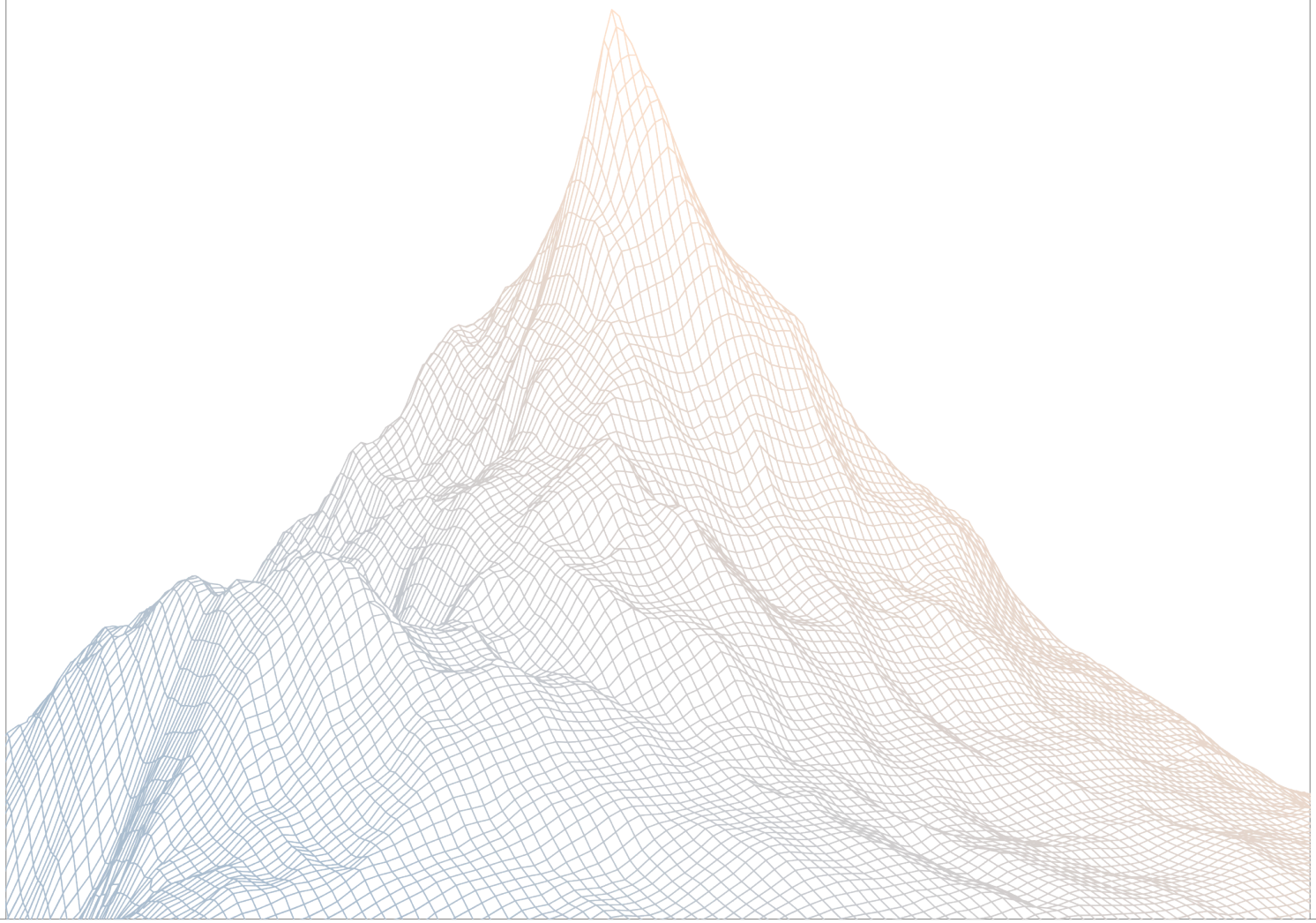
Zenith

Spectral

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: February 17th to February 26th, 2025

AUDITED BY: Matte
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	`be2130f1279345e2e829bb0d92e3bc11fba3e8ef`
Files	`contracts/octoDistributor/OctoDistributor.sol`

2.3 Audit Timeline

February 17, 2025	Audit start
February 26, 2025	Audit end
March 03, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	2
Medium Risk	2
Low Risk	1
Informational	2
Total Issues	**7**

3

Findings Summary

ID	Description	Status
H-1	Incorrect Token Symbol Check Leads to Bypassing Special Distribution Logic	Resolved
H-2	The bonus amount of Agent tokens is not calculated correctly	Resolved
M-1	The OctoDistributor doesn't work when the agent token is set to address(0)	Resolved
M-2	The withdrawAllAgentTokens function works incorrectly in OctoDistributor	Resolved
L-1	An additional check is needed in the setParameters function of the OctoDistributor	Resolved
I-1	Missing ANS Node Address Validation Can Lead to Permanent Loss of User Funds	Resolved
I-2	There could be some dust amounts in the transferTradingRewards function	Resolved

4

Findings

4.1 High Risk

A total of 2 high risk findings were identified.

[H-1] Incorrect Token Symbol Check Leads to Bypassing Special Distribution Logic

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

Target

- OctoDistributor.sol

Description:

The `transferTradingRewards` function checks for the token symbol "SPECTRA" to determine if special distribution logic should be applied:

```
if(keccak256(abi.encodePacked("SPECTRA")) ==
    keccak256(abi.encodePacked(IAgentToken(agentToken).symbol())))
```

However, the actual Spectral token (0x96419929d7949D6A801A6909c145C8EEf6A40431) has the symbol "SPEC". This mismatch means the special distribution logic (20% treasury, 20% spectra wallet, 60% employees) is never triggered, and instead falls back to the generic distribution (20% treasury, 80% creator).

This not only causes incorrect token distribution but also opens up potential manipulation as anyone could create a token with symbol "SPECTRA" to trigger the special distribution logic.

Recommendations:

Replace the symbol comparison with a direct address check:

```
if(agentToken == address(spectral_token)) {
    // Special distribution logic
```

```
}
```

Spectral: Resolved with @0d21cb9101b...

Zenith: Verified.

[H-2] The bonus amount of Agent tokens is not calculated correctly

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

Target

- AgenticCompany.sol

Description:

When transferring the `Agent token` to the contract, a `tax` is applied because it is a `fee-on-transfer` token. AgentToken.sol#L94

```
function transferFrom(address sender, address recipient, uint256 amount)
    public override returns (bool) {
    ...
    if (recipient.isContract()) {
        uint256 taxAmount = (amount * taxPercentage) / 10000;
        uint256 amountAfterTax = amount - taxAmount;

        super.transferFrom(sender, taxWallet, taxAmount/2);
        super.transferFrom(sender, address(this), taxAmount/2);
        agentBalances.deposit(address(this), address(this), address(this),
    taxAmount/2);

        return super.transferFrom(sender, recipient, amountAfterTax);
    }
}
```

However, this `tax` is not considered when the `job manager deposits bonuses` for the job. When the `agentTokenAmount` of `Agent token` is transferred to the `AgenticCompany` contract, the actual transferred amount is less than the `agentTokenAmount` due to the `tax`. Despite this, the `hiringBonusAgentToken` value increases by the full `agentTokenAmount` regardless of the `tax`. AgenticCompany.sol#L553-L556

```
function _depositHiringBonuses(
    AgenticCompanyStorage storage $,
```

```
    Job storage job,
    uint256 spectralAmount,
    uint256 agentTokenAmount,
    uint256 usdcAmount
) private {

    if (agentTokenAmount > 0) {
        job.hiringBonusAgentToken += agentTokenAmount;
        IERC20($.agentToken).safeTransferFrom(_msgSender(), address(this),
    agentTokenAmount);
    }

}
```

As a result, there are insufficient tokens in the `AgenticCompany` contract when distributing `bonuses`.

This issue is also not considered in the `transferHiringDistributions` function of the `OctoDistributor`. OctoDistributor.sol#L325

```
function transferHiringDistributions(
    HiringDistribution[] calldata distributions,
    address agentToken,
    uint256 totalSpec,
    uint256 totalAgentToken,
    uint256 totalUsdc
) external nonReentrant onlyAgenticCompany() {
    require(agentToken ≠ address(0), "ZERO_ADDRESS");
    IERC20Upgradeable(agentToken).safeTransferFrom(msg.sender,
    address(this), totalAgentToken);
    for (uint256 i = 0; i < distributions.length; i++) {
        UserBalances storage user = userBalances[

    systemAddresses.ansResolver.addr(distributions[i].recipientAnsNode)
        ];
        user.agent_tokens_list[agentToken]
    += distributions[i].agentTokenAmount;
        accAgentTokenAmount += distributions[i].agentTokenAmount;
    }
    require(accAgentTokenAmount == totalAgentToken,
    "INCORRECT_AGENT_TOKEN_AMOUNT");
}
```

Recommendations:

```
function _depositHiringBonuses(
    AgenticCompanyStorage storage $,
    Job storage job,
    uint256 spectralAmount,
    uint256 agentTokenAmount,
    uint256 usdcAmount
) private {

    if (agentTokenAmount > 0) {
+^^I^^I uint256 prevBalance = IERC20($.agentToken).balanceOf(address(this));
-         job.hiringBonusAgentToken += agentTokenAmount;
        IERC20($.agentToken).safeTransferFrom(_msgSender(), address(this),
    agentTokenAmount);
+ ^^I^^I job.hiringBonusAgentToken
    += IERC20($.agentToken).balanceOf(address(this)) - prevBalance;
    }

}
```

Fixing the `transferHiringDistributions` function could become more difficult. One possible solution is outlined below, but it does not take the `dust` amount into account.

```
function transferHiringDistributions(
    HiringDistribution[] calldata distributions,
    address agentToken,
    uint256 totalSpec,
    uint256 totalAgentToken,
    uint256 totalUsdc
) external nonReentrant onlyAgenticCompany() {
    require(agentToken ≠ address(0), "ZERO_ADDRESS");

+^^I uint256 prevBalance
    = IERC20Upgradeable(agentToken).balanceOf(address(this));

    IERC20Upgradeable(agentToken).safeTransferFrom(msg.sender,
    address(this), totalAgentToken);

+    uint256 receivedAgentToken
    = IERC20Upgradeable(agentToken).balanceOf(address(this)) - prevBalance;

    for (uint256 i = 0; i < distributions.length; i++) {
        UserBalances storage user = userBalances[
```

```
    systemAddresses.ansResolver.addr(distributions[i].recipientAnsNode)
        ];

-^^I^^I user.agent_tokens_list[agentToken]
    += distributions[i].agentTokenAmount;
+        user.agent_tokens_list[agentToken] += totalAgentToken == 0 ? 0 :
    distributions[i].agentTokenAmount * receivedAgentToken
    / totalAgentToken;

        accAgentTokenAmount += distributions[i].agentTokenAmount;
    }
    require(accAgentTokenAmount == totalAgentToken,
    "INCORRECT_AGENT_TOKEN_AMOUNT");
}
```

Spectral: Resolved with @2c092db2d3... & @6e6c23f2fdc...

Zenith: Verified.

4.2 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] The OctoDistributor doesn't work when the agent token is set to address(0)

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

4.3 Target

- OctoDistributor.sol

Description:

The `agent token` can be `address(0)` by design in the `AgenticCompany`. For example, when the `company` is created in the `factory`, the validity of the `agent token` is checked only when it is not `address(0)`. AgenticCompanyFactory.sol#L64

```
function createCompany(string calldata companyName, address agentToken)
    external returns (address companyAddress_) {
    AgenticCompanyFactoryStorage storage $
    = _getAgenticCompanyFactoryStorage();
    if (agentToken ≠ address(0) && $.allowOnlyValidAgentTokens) {
        _checkValidAgentToken(agentToken);
    }
}
```

This means there are cases where the `agent token` can be `address(0)`.

A stricter check occurs in the `_dissolveCompany` function of the `AgenticCompany`. At `line 606`, the `agent token balance` is calculated based on whether the `agent token` is `address(0)` or not. AgenticCompany.sol#L606

```
function _dissolveCompany(AgenticCompanyStorage storage $,
    address withdrawTo) private {
```

```
    $.isDissolved = true;

    address agentToken = $.agentToken;

    uint256 specBalance = IERC20(SPECTRAL_TOKEN).balanceOf(address(this));

606:    uint256 agentTokenBalance = agentToken == address(0) ? 0 :
    IERC20(agentToken).balanceOf(address(this));

    uint256 usdcBalance = IERC20(USDC).balanceOf(address(this));
}
```

In short, the `company` is compatible with an `empty agent token`.

However, in the `_distributeHiringBonuses` function, the `transferHiringDistributions` function of the `distributor` is called. AgenticCompany.sol#L517-L523

```
function _distributeHiringBonuses(
    AgenticCompanyStorage storage $,
    uint256 jobHiringBonusSpec,
    uint256 jobHiringBonusAgentToken,
    uint256 jobHiringBonusUsdc,
    bytes32 hiredApplicantAnsNode,
    bytes32[] calldata shortlistAnsNodes
) private {

    IDistributor(DISTRIBUTOR).transferHiringDistributions(
        distributions,
        $.agentToken,
        totalSpecDistributed,
        totalAgentTokenDistributed,
        totalUsdcDistributed
    );

}
```

In the `transferHiringDistributions` function, the check at `line 322` causes the transaction to be reverted. OctoDistributor.sol#L322

```
function transferHiringDistributions(
    HiringDistribution[] calldata distributions,
    address agentToken,
    uint256 totalSpec,
    uint256 totalAgentToken,
    uint256 totalUsdc
```

```
) external nonReentrant onlyAgenticCompany() {
    require(
        distributions.length > 0,
        "DISTRIBUTIONS_MUST_BE_PROVIDED"
    );
322:    require(agentToken ≠ address(0), "ZERO_ADDRESS");
}
```

The `OctoDistributor` should also be compatible with an `empty agent token`.

Recommendations:

It is possible that the intended design of the `OctoDistributor` does not allow an `empty agent token`. However, this issue occurs due to the interaction of multiple contracts. I have found a solution to modify the `transferHiringDistributions` function as follows:

```
function transferHiringDistributions(
    HiringDistribution[] calldata distributions,
    address agentToken,
    uint256 totalSpec,
    uint256 totalAgentToken,
    uint256 totalUsdc
) external nonReentrant onlyAgenticCompany() {
    require(
        distributions.length > 0,
        "DISTRIBUTIONS_MUST_BE_PROVIDED"
    );
-    require(agentToken ≠ address(0), "ZERO_ADDRESS");
    usdc_token.safeTransferFrom(msg.sender, address(this), totalUsdc);
    spectral_token.safeTransferFrom(msg.sender, address(this), totalSpec);
+^^I if (agentToken ≠ address(0) && totalAgentToken ≠ 0) {
        IERC20Upgradeable(agentToken).safeTransferFrom(msg.sender,
    address(this), totalAgentToken);
+^^I }

    uint256 accSpecAmount = 0;
    uint256 accAgentTokenAmount = 0;
    uint256 accUsdcAmount = 0;
    for (uint256 i = 0; i < distributions.length; i++) {
        UserBalances storage user = userBalances[

    systemAddresses.ansResolver.addr(distributions[i].recipientAnsNode)
        ];
        user.spectral += distributions[i].specAmount;
        accSpecAmount += distributions[i].specAmount;
```

```
+^^I^^I if (agentToken ≠ address(0)) {
            _addOrGetAgentToken(user, agentToken);
            user.agent_tokens_list[agentToken]
    += distributions[i].agentTokenAmount;
+^^I^^I }

        accAgentTokenAmount += distributions[i].agentTokenAmount;
        user.usdc += distributions[i].usdcAmount;
        accUsdcAmount += distributions[i].usdcAmount;
    }
}
```

Spectral: Resolved with @ad6b4e63ea5...

Zenith: Verified

[M-2] The withdrawAllAgentTokens function works incorrectly in OctoDistributor

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

Target

- OctoDistributor.sol

Description:

Suppose a user's `agent_tokens` are `[A, B, C, D]`, and they want to `withdraw` `C` and `D`. The `start_index` is `2` and the `end_index` is `4`. The `withdrawal` of `C` will succeed, and this token is removed immediately at `line 223` before `withdrawing` `D`. OctoDistributor.sol#L223

```
function withdrawAllAgentTokens(uint256 start_index, uint256 end_index)
    external nonReentrant {
    require(start_index < end_index, "INVALID_INDEX");
    require(end_index <= userBalances[msg.sender].agent_tokens.length,
    "INVALID_INDEX");
    for (uint256 i = start_index; i < end_index; i++) {
        address token = userBalances[msg.sender].agent_tokens[i];
        uint256 amount = userBalances[msg.sender].agent_tokens_list[token];
        require(amount > 0, "AMOUNT_ZERO");
        userBalances[msg.sender].agent_tokens_list[token] = 0;
223:        _removeAgentToken(userBalances[msg.sender], token);
        IERC20Upgradeable(token).safeTransfer(msg.sender, amount);
        emit Withdraw(msg.sender, token, amount);
    }
}
```

In the `_removeAgentToken` function, `C` is replaced with `D`, so the `agent_tokens` become `[A, B, D]`. OctoDistributor.sol#L308

```
function _removeAgentToken(UserBalances storage user, address token)
    internal {
    uint256 index = user.agent_tokens_indecies[token];
    require(index < user.agent_tokens.length, "Token not found");
```

```
    address lastToken = user.agent_tokens[user.agent_tokens.length - 1];

    user.agent_tokens[index] = lastToken;
    user.agent_tokens_indecies[lastToken] = index;

    delete user.agent_tokens_indecies[token];

    user.agent_tokens.pop();
}
```

Returning to the `withdrawAllAgentTokens` function, when the `index i` is 3, it is out of the `agent_tokens` array, causing the transaction to revert. Users do not know the reason for the revert.

Recommendations:

The removal should be done after withdrawing all tokens. Additionally, please remember that the `index` should be processed from right to left.

```
function withdrawAllAgentTokens(uint256 start_index, uint256 end_index)
    external nonReentrant {
    require(start_index < end_index, "INVALID_INDEX");
    require(end_index <= userBalances[msg.sender].agent_tokens.length,
    "INVALID_INDEX");
    for (uint256 i = start_index; i < end_index; i++) {
        address token = userBalances[msg.sender].agent_tokens[i];
        uint256 amount = userBalances[msg.sender].agent_tokens_list[token];
        require(amount > 0, "AMOUNT_ZERO");
        userBalances[msg.sender].agent_tokens_list[token] = 0;
-        _removeAgentToken(userBalances[msg.sender], token);
        IERC20Upgradeable(token).safeTransfer(msg.sender, amount);
        emit Withdraw(msg.sender, token, amount);
    }

+^^Ifor (uint256 i = end_index - 1; i >= start_index; ) {
+^^I^^I_removeAgentToken(userBalances[msg.sender],
    userBalances[msg.sender].agent_tokens[i]);
+^^I^^Iif (i == 0) {
+^^I^^I^^Ibreak;
+^^I^^I}
+^^I^^Ii --;
+^^I}
}
```

Spectral: Resolved with @ab802f900cd3...

Zenith: Verified.

4.4 Low Risk

A total of 1 low risk findings were identified.

[L-1] An additional check is needed in the setParameters function of the OctoDistributor

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

Target

- OctoDistributor.sol

Description:

The initial values of `parameters` are listed below. OctoDistributor.sol#L158-L161

```
function initialize(
    address _autonomousDeployer,
    address _agenticCompanyFactory,
    address _ansResolver,
    address _ansReverseRegistrar,
    address _spectralTreasury,
    uint256 _spectraCompanyIndex,
    address _usdc_token,
    address _spectral_token
) public initializer {
    parameters[Parameter.TRADING_REWARDS_TREASURY_CUT] = 2000;
    parameters[Parameter.TRADING_REWARDS_CREATOR_CUT] = 8000;
    parameters[Parameter.TRADING_REWARDS_SPECTRA_CUT] = 2000;
    parameters[Parameter.TRADING_REWARDS_EMPLOYEES_CUT] = 6000;
}
```

There should be a strict relationship between these values like below:

```
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;
```

This ensures the correct token transfer in the `transferTradingRewards` function.

However, in the `setParameters` function, there is no such check. OctoDistributor.sol#L210

```
function setParameters(
    Parameter[] calldata _parameters,
    uint256[] calldata _values
) external onlyOwner {
    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]);
    }
}
```

Recommendations:

Although this function can only be called by the owner, it would be better to add this check.

Spectral: Resolved with @185f2d97582...

Zenith: Verified.

4.5 Informational

A total of 2 informational findings were identified.

[I-1] Missing ANS Node Address Validation Can Lead to Permanent Loss of User Funds

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

Target

- OctoDistributor.sol

Description:

When distributing hiring bonuses or trading rewards, the `OctoDistributor` contract looks up recipient addresses using their ANS nodes. If a recipient hasn't set their address in the ANS resolver, the funds will be credited to `address(0)` and become permanently locked.

This happens in two key places:

1. During hiring bonus distribution:

```
File: OctoDistributor.sol

311:     function transferHiringDistributions(
312:         HiringDistribution[] calldata distributions,
313:         address agentToken,
314:         uint256 totalSpec,
315:         uint256 totalAgentToken,
316:         uint256 totalUsdc
317:     ) external nonReentrant onlyAgenticCompany() {
318:         require(
319:             distributions.length > 0,
320:             "DISTRIBUTIONS_MUST_BE_PROVIDED"
321:         ); // ...
322:
329:         for (uint256 i = 0; i < distributions.length; i++) {
```

```
330:             UserBalances storage user = userBalances[
331:   >>           systemAd-
    dresses.ansResolver.addr(distributions[i].recipientAnsNode) //
332:             ];
                // ...
```

2. During trading rewards distribution to employees:

```
File: OctoDistributor.sol

240:     function transferTradingRewards(
241:         address agentToken,
242:         uint256 usdcAmount
243:     ) external nonReentrant onlyAdmin() {
244:
                        //...
260:             for (uint256 i = 0; i < employeeCount; i++) {
261:                 address employee =
262: >>                systemAd-
    dresses.ansResolver.addr(IAgenticCompany(systemAddresses.agenticCompanyFactory.g
263:                 userBalances[employee].usdc += perEmployeeAmount;
                        //..
```

Unlike the AgenticCompany contract which has a check in `_getAnsAddress()`:

```
File: AgenticCompany.sol
653:     function _getAnsAddress(bytes32 ansNode)
    private view returns (address addr_) {
654:         addr_ = IANSResolver(ANS_RESOLVER).addr(ansNode);
655:         if (addr_ == address(0)) {
656:             revert AnsNodeMustResolveToAddress(ansNode);
657:         }
658:     }
```

The `OctoDistributor` has no such validation, allowing funds to be credited to `address(0)` if the ANS node doesn't resolve to an address.

Once funds are credited to `userBalances[address(0)]`, they become permanently locked.

This path is valid only after `debitInterviewFees` call.

Recommendations:

Add validation in the `OctoDistributor` contract too preferably inlined in both functions.

Spectral: Resolved with @e326175b0e...

Zenith: Verified

[I-2] There could be some dust amounts in the transferTradingRewards function

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

Target

- OctoDistributor.sol

Description:

In the `transferTradingRewards` function, `USDC` is transferred to several parties, and the individual amounts are calculated according to their percentages. As a result, there are some dust amounts of `USDC` left after the transfer.

```
function transferTradingRewards(
    address agentToken,
    uint256 usdcAmount
) external nonReentrant onlyAdmin() {
    require(agentToken ≠ address(0), "ZERO_ADDRESS");
    require(usdcAmount > 0, "AMOUNT_ZERO");
    address[] memory beneficiaries;
    uint256[] memory amounts;
    usdc_token.safeTransferFrom(msg.sender, address(this), usdcAmount);
    uint256 treasuryAmount = (usdcAmount
    * parameters[Parameter.TRADING_REWARDS_TREASURY_CUT]) / PRECISION;
    usdc_token.safeTransfer(systemAddresses.spectralTreasury,
    treasuryAmount);
    if(keccak256(abi.encodePacked("SPECTRA")) ==
    keccak256(abi.encodePacked(IAgentToken(agentToken).symbol()))) {
        uint256 spectraAmount = (usdcAmount
    * parameters[Parameter.TRADING_REWARDS_SPECTRA_CUT]) / PRECISION;
        uint256 employeesAmount = (usdcAmount
    * parameters[Parameter.TRADING_REWARDS_EMPLOYEES_CUT]) / PRECISION;
        address spectraWallet =
    IAgentBalances(systemAddresses.autonomousDeployer.agentBalances()).agentWallets(a
        uint256 employeeCount =
    IAgenticCompany(systemAddresses.agenticCompanyFactory.getCompanyAddressAtIndex(sp
        require(employeeCount > 0, "NO_EMPLOYEES");
        beneficiaries = new address[](employeeCount);
```

```
        amounts = new uint256[](employeeCount);
        uint256 perEmployeeAmount = employeesAmount / employeeCount;
        for (uint256 i = 0; i < employeeCount; i++) {
            address employee =

    systemAddresses.ansResolver.addr(IAgenticCompany(systemAddresses.agenticCompanyFa
            userBalances[employee].usdc += perEmployeeAmount;
            beneficiaries[i] = employee;
            amounts[i] = perEmployeeAmount;
        }
        usdc_token.safeTransfer(spectraWallet, spectraAmount);
        emit DistributeSpectraTradingRewards(agentToken, usdcAmount,
    beneficiaries, amounts);
    } else {
        uint256 creatorAmount = (usdcAmount
    * parameters[Parameter.TRADING_REWARDS_CREATOR_CUT]) / PRECISION;
        require(agentCreators[agentToken] ≠ address(0), "CREATOR_NOT_SET");
        usdc_token.safeTransfer(agentCreators[agentToken], creatorAmount);
        emit DistributeGenericTradingRewards(agentToken, usdcAmount);
    }
}
```

Recommendations:

The last transfer amount can be calculated as the difference between the total transfer amount and the amounts transferred to other parties previously.

Client: Resolved with @c68422d585916... & @25c222033558...

Zenith: Verified