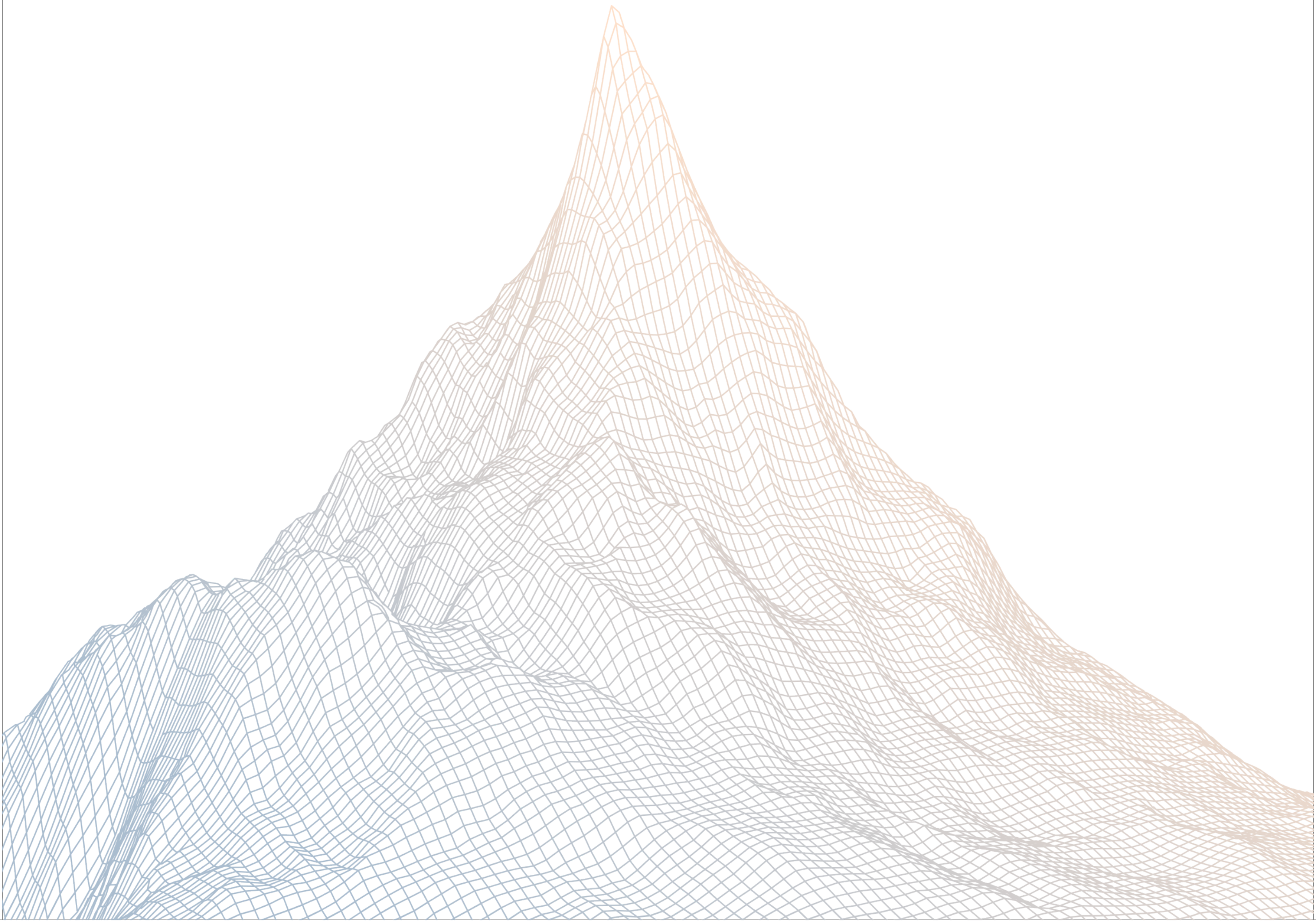
Zenith

Spectral

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: July 21th to July 23th, 2025

AUDITED BY: SorryNotSorry
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 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	`a818c081e67f31b9e66207af71f812c2ae880483`
Files	`Changes in PR-38`

2.3 Audit Timeline

July 21, 2025	Audit start
July 23, 2025	Audit end
August 5, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	3
Medium Risk	5
Low Risk	2
Informational	5
Total Issues	**15**

3 Findings Summary

ID	Description	Status
H-1	Refunded agent tokens should be sent back to the requester	Resolved
H-2	The tax in agent tokens is causing incorrect accounting in the AgentImageService	Resolved
H-3	Wrong approval in fullfillImage	Resolved
M-1	Duplicate requestId values can occur	Resolved
M-2	The requestRefund function contains an incorrect check	Resolved
M-3	Swap deadline in fullfillImage might stall the TX	Resolved
M-4	The configuration cannot be changed once it has been set	Resolved
M-5	The owner cannot set a new creator for the agent tokens	Acknowledged
L-1	setDistributor and setSpectralStaking functions might be needed	Acknowledged
L-2	The fees can be transferred to the admin	Resolved
I-1	Unused mapping, variable and events in V2	Resolved
I-2	Recommendation for public mempool chain deployments	Acknowledged
I-3	Unused variables	Resolved
I-4	The AgentImageService should inherit from IAgentImageService	Resolved
I-5	The previously calculated value can be reused	Resolved

4 Findings

4.1 High Risk

A total of 3 high risk findings were identified.

[H-1] Refunded agent tokens should be sent back to the requester

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

Target

- AgentImageService.sol

Description:

When users request an image, they send `agent tokens` and are registered as the `requesters`.

- AgentImageService.sol#L203

```
function requestImage(
    address agentToken,
    string calldata prompt,
    uint256 width,
    uint256 height
) external nonReentrant onlyValidAgent(agentToken) {
    token.safeTransferFrom(msg.sender, address(this), finalImagePrice);

    imageRequests[agentToken][params.requestId] = ImageRequest({
        user: msg.sender,
        agent: agentToken,
        amount: finalImagePrice,
        width: width,
        height: height,
        timestamp: block.timestamp,
        fulfilled: 0,
        refundRequested: 0,
        refunded: 0,
        prompt: prompt
    });
}
```

They may later request a `refund` for certain images. [AgentImageService.sol#L263]

```
function requestRefund(
    address agentToken,
    bytes32 requestId
) external nonReentrant {
    ImageRequest storage request = imageRequests[agentToken][requestId];
    AgentConfig memory config = agentConfigs[agentToken];

    require(request.user == msg.sender, "Not request owner");
    require(request.fulfilled == 0 && request.refunded == 0 &&
    request.refundRequested == 0, "Request already processed");
    require(
        block.timestamp >= request.timestamp + config.refundTimeLimit,
        "Refund time limit not reached"
    );
@->    request.refundRequested = block.timestamp;
    emit RefundRequested(agentToken, requestId, request.amount);
    // If the request was not fulfilled and the refund time limit has passed,
    issue a refund automatically
    if(request.fulfilled == 0 && block.timestamp >= (request.timestamp
    + config.refundTimeLimit)) {
        issueRefund(agentToken, requestId, request);
    }
}
```

If the `owner` of the `agent token` or an `admin` approves the `refund`, the `issueRefund` function is called. [AgentImageService.sol#L284]

```
function approveRefund(
    address agentToken,
    bytes32 requestId
) external onlyAgentOwner(agentToken) nonReentrant {
    issueRefund(agentToken, requestId, request);
}
```

However, in this case, `msg.sender` is not the `original requester`. [AgentImageService.sol#L299]

```
function issueRefund(
    address agentToken,
    bytes32 requestId,
    ImageRequest storage request
) internal {
    // Transfer tokens back to user
    IERC20Upgradeable(agentToken).safeTransfer(msg.sender, request.amount);
}
```

The `refunded tokens` should obviously be sent to the `original requester`, not the caller.

Recommendations:

```
function issueRefund(
    address agentToken,
    bytes32 requestId,
    ImageRequest storage request
) internal {
    // Transfer tokens back to user
    IERC20Upgradeable(agentToken).safeTransfer(msg.sender, request.amount);

     IERC20Upgradeable(agentToken).safeTransfer(request.user, request.amount);
}
```

Spectral: Resolved with @8c5739a514a93...

Zenith: Verified.

[H-2] The tax in agent tokens is causing incorrect accounting in the AgentImageService

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

Target

- AgentImageService.sol

Description:

When `agent tokens` are transferred, a portion is sent to the `deployer` as a `tax`.

- AgentToken.sol#L100

```
function transfer(address recipient, uint256 amount)
    public override returns (bool) {
    if (recipient == address(agentBalances)) {
        return super.transfer(recipient, amount);
    }
    address distributor = address(deployer.distributor());
    if (address(deployer) != address(0) &&                          //
    deployer exists
    (recipient.isContract() || msg.sender.isContract())
    &&                    // XOR: exactly one must be a contract
    !(recipient == address(deployer) || msg.sender == address(deployer))
    &&  // neither sender nor recipient is deployer
    (distributor == address(0) || recipient != distributor))
    {
        uint256 taxAmount = (amount * taxPercentage) / 10000;
        uint256 amountAfterTax = amount - taxAmount;

        //send the tax to deployer to distribute
@->        super.transfer(address(deployer), taxAmount);
        deployer.accumulateSwapFees(taxAmount);

        // Transfer the remaining amount to the recipient
        return super.transfer(recipient, amountAfterTax);
    } else {
        // No tax if recipient is not a contract
        return super.transfer(recipient, amount);
    }
}
```

However, this `tax` is not accounted for in the `AgentImageService`.

- AgentImageService.sol#L216

```
function requestImage(
    address agentToken,
    string calldata prompt,
    uint256 width,
    uint256 height
) external nonReentrant onlyValidAgent(agentToken) {
    token.safeTransferFrom(msg.sender, address(this), finalImagePrice);

    // Add to pending fees instead of accumulated
    pendingFees[agentToken] += finalImagePrice;
}
```

As a result, `pendingFees` reflects a higher amount than what was actually received.

Recommendations:

```
function requestImage(
    address agentToken,
    string calldata prompt,
    uint256 width,
    uint256 height
) external nonReentrant onlyValidAgent(agentToken) {
    uint256 prevBalance = token.balanceOf(address(this));

    token.safeTransferFrom(msg.sender, address(this), finalImagePrice);

    finalImagePrice = token.balanceOf(address(this)) - prevBalance;

    // Add to pending fees instead of accumulated
    pendingFees[agentToken] += finalImagePrice;
}
```

Alternatively, remove the `tax` when the recipient is the `AgentImageService` in `Agent token` transfers.

Spectral: Resolved with @7043302ad...

Zenith: Verified.

[H-3] Wrong approval in fullfillImage

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

Target

- AgentImageService.sol

Description:

In `fulfillImage` the swap path is coded as;

```
uint256 oldSpecBalance
    = IERC20Upgradeable(deployer.spectralToken()).balanceOf(address(this));
--
IERC20Upgradeable(agentToken).approve(spectralTreasury, treasuryCut);
uint256 specAmount = deployer.getSPECAmountForTokens(treasuryCut,
    agentToken);
```

Inside `swapExactTokensForSPEC`, the deployer contract does ;

```
IERC20Upgradeable(fromToken).safeTransferFrom(msg.sender, address(this),
    amountIn);
```

where `msg.sender` is the `AgentImageService` and `address(this)` is the `deployer`

In order for the `safeTransferFrom` to succeed, the contract needs to have given the `deployer` allowance, not the `treasury`

Recommendations:

```
IERC20Upgradeable(agentToken).approve(spectralTreasury, treasuryCut);
IERC20Upgradeable(agentToken).approve(deployer, treasuryCut);
```

Spectral: Resolved with @8996a4343...

Zenith: Verified.

4.2 Medium Risk

A total of 5 medium risk findings were identified.

[M-1] Duplicate requestId values can occur

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

Target

- AgentImageService.sol

Description:

When users request an `image`, the `requestId` is generated as shown below.

- AgentImageService.sol#L191

```
function requestImage(
    address agentToken,
    string calldata prompt,
    uint256 width,
    uint256 height
) external nonReentrant onlyValidAgent(agentToken) {
    // Generate request ID from prompt and timestamp
    RequestParams memory params = RequestParams({
        requestId:
     keccak256(abi.encodePacked(prompt, block.timestamp, msg.sender)),
        tokenPriceFromSpec:
        deployer.getTokensReceived
        (parameters[Parameter.GLOBAL_MIN_SPEC_PRICE_PER_IMAGE],
        agentToken)
    });

    token.safeTransferFrom(msg.sender, address(this), finalImagePrice);

    // Record the request
    imageRequests[agentToken][params.requestId] = ImageRequest({
        user: msg.sender,
        agent: agentToken,
        amount: finalImagePrice,
        width: width,
        height: height,
        timestamp: block.timestamp,
        fulfilled: 0,
        refundRequested: 0,
        refunded: 0,
        prompt: prompt
    });
}
```

This approach allows for the possibility that if the same user makes two `requests` with the same `prompt` within the same block, the first `request` may be ignored. However, the tokens for both `requests` are already sent. As a result, the tokens associated with the first `request` would be lost.

Recommendations:

```
function requestImage(
    address agentToken,
    string calldata prompt,
    uint256 width,
    uint256 height
) external nonReentrant onlyValidAgent(agentToken) {
    // Generate request ID from prompt and timestamp
    RequestParams memory params = RequestParams({
        requestId: keccak256
        (abi.encodePacked(prompt, block.timestamp, msg.sender)),
        tokenPriceFromSpec:
         deployer.getTokensReceived
         (parameters
         [Parameter.GLOBAL_MIN_SPEC_PRICE_PER_IMAGE],
         agentToken)
    });

    token.safeTransferFrom
    (msg.sender, address(this), finalImagePrice);

    require(imageRequests[agentToken]
   [params.requestId].user == address(0), "");

    // Record the request
    imageRequests
```

```
      [agentToken][params.requestId] = ImageRequest({
          user: msg.sender,
          agent: agentToken,
          amount: finalImagePrice,
          width: width,
          height: height,
          timestamp: block.timestamp,
          fulfilled: 0,
          refundRequested: 0,
          refunded: 0,
          prompt: prompt
      });
  }
```

Spectral: Resolved with @6a9d6d70de...

Zenith: Verified.

[M-2] The requestRefund function contains an incorrect check

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

Target

- AgentImageService.sol

Description:

Users can request `refunds` for certain `images` using the `requestRefund` function.

- AgentImageService.sol#L260

```
function requestRefund(
    address agentToken,
    bytes32 requestId
) external nonReentrant {
    ImageRequest storage request = imageRequests[agentToken][requestId];
    AgentConfig memory config = agentConfigs[agentToken];

    require(request.user == msg.sender, "Not request owner");
258:require(request.fulfilled == 0 && request.refunded == 0 &&
    request.refundRequested == 0, "Request already processed");
    require(
260:    block.timestamp >= request.timestamp + config.refundTimeLimit,
        "Refund time limit not reached"
    );
    request.refundRequested = block.timestamp;
    emit RefundRequested(agentToken, requestId, request.amount);
    // If the request was not fulfilled and the refund time limit has passed,
    issue a refund automatically
266:if(request.fulfilled == 0 && block.timestamp >= (request.timestamp
    + config.refundTimeLimit)) {
        issueRefund(agentToken, requestId, request);
    }
}
```

`Line 258` checks that the `image` has not been `fulfilled` or `refunded` yet. `Line 260` checks that the configured `refundTimeLimit` has passed. Due to these two checks, the check at `line 266` always evaluates to `true`, causing the `refund` to be executed immediately upon the `request`.

However, as the name implies, `config.refundTimeLimit` is intended to define when the actual _refund_ can be processed — not when the _refund request_ can be made.

Users should be allowed to request a `refund` at any time, but the actual `refund` should only be executed after the time limit has passed. Therefore, the check in `line 260` is unnecessary and can be safely removed.

Recommendations:

```
function requestRefund(
    address agentToken,
    bytes32 requestId
) external nonReentrant {
    ImageRequest storage request = imageRequests[agentToken][requestId];
    AgentConfig memory config = agentConfigs[agentToken];

    require(request.user == msg.sender, "Not request owner");
    require(request.fulfilled == 0 && request.refunded == 0 &&
    request.refundRequested == 0, "Request already processed");
    require(
         block.timestamp >= request.timestamp + config.refundTimeLimit,
        "Refund time limit not reached"
    );
    request.refundRequested = block.timestamp;
    emit RefundRequested(agentToken, requestId, request.amount);
    // If the request was not fulfilled and the refund time limit has passed,
    issue a refund automatically
    if(request.fulfilled == 0 && block.timestamp >= (request.timestamp
    + config.refundTimeLimit)) {
        issueRefund(agentToken, requestId, request);
    }
}
```

Spectral: Resolved with @7ee52d8a8...

Zenith: Verified.

[M-3] Swap deadline in fullfillImage might stall the TX

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

Target

- AgentImageService.sol

Description:

The `fulfilImage` function is as below;

```
function fulfillImage(
--
address agentToken,
bytes32 requestId
) external onlyAgentOwner(agentToken) onlyValidAgent(agentToken)
    nonReentrant {
ImageRequest storage request = imageRequests[agentToken][requestId];
require(request.fulfilled == 0 && request.refunded == 0, "Request already
    processed");
require(request.user ≠ address(0), "Request does not exist");

request.fulfilled = block.timestamp;

// Move fee from pending to accumulated
pendingFees[agentToken] -= request.amount;
uint256 treasuryCut =
(request.amount * parameters[Parameter.TREASURY_CUT]) / 10000;
accumulatedFees[agentToken] += (request.amount - treasuryCut);
uint256 oldSpecBalance =
IERC20Upgradeable(deployer.spectralToken()).balanceOf(address(this));
IERC20Upgradeable(agentToken).approve(spectralTreasury, treasuryCut);
uint256 specAmount =
 deployer.getSPECAmountForTokens(treasuryCut, agentToken);
deployer.swapExactTokensForSPEC
(treasuryCut, (specAmount * 95) / 100,
 agentToken, block.timestamp + 20 minutes); <<<
uint256 newSpecBalance =
 IERC20Upgradeable(deployer.spectralToken())
 .balanceOf(address(this));
IERC20Upgradeable(deployer.spectralToken())
.safeTransfer(spectralTreasury, newSpecBalance - oldSpecBalance);
emit ImageFulfilled(agentToken, requestId);
}
```

The `deadline` for the swap param is defined as `block.timestamp + 20 minutes` However, this prolonged deadline could end up with the TX being stalled for longer periods which troubles the smooth operation.

Recommendations:

```diff
- deployer.swapExactTokensForSPEC(treasuryCut, (specAmount * 95) / 100,
-     agentToken, block.timestamp + 20 minutes);
+ deployer.swapExactTokensForSPEC(treasuryCut, (specAmount * 95) / 100,
+     agentToken, block.timestamp);
```

Spectral: Resolved with @353e274ccb...

Zenith: Verified.

[M-4] The configuration cannot be changed once it has been set

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

Target

- AgentImageService.sol

Description:

When an `agent` is created using `deployImageAgent`, certain constant values are set as its `configuration`.

- AgentImageService.sol#L137

```
function deployImageAgent(
    string memory agentName,
    string memory agentTicker,
    string memory agentDescription,
    address owner,
    uint256 minSpecAmount) external payable returns (address agentToken) {
    agentConfigs[agentToken] = AgentConfig({
        pricePerImage: 1 ether,
        imageDescription: "",
        refundTimeLimit: 1 days,
        isConfigured: true
    });
}
```

After this, the `configuration` cannot be updated.

- AgentImageService.sol#L171

```
function configureAgent(
    address agentToken,
    uint256 pricePerImage,
    string memory imageDescription,
    uint256 refundTimeLimit
) external onlyAgentOwner(agentToken) onlyValidAgent(agentToken) {
    require(!agentConfigs[agentToken].isConfigured, "Agent already
    configured");
    _configureAgent(agentToken, pricePerImage, imageDescription,
    refundTimeLimit);

    emit AgentConfigured(agentToken, pricePerImage, imageDescription,
    refundTimeLimit);
}
```

Allowing the `creator` to update the `configuration` would provide greater flexibility.

Recommendations:

```
function configureAgent(
    address agentToken,
    uint256 pricePerImage,
    string memory imageDescription,
    uint256 refundTimeLimit
) external onlyAgentOwner(agentToken) onlyValidAgent(agentToken) {
    require(!agentConfigs[agentToken].isConfigured,
       "Agent already configured");
    _configureAgent(agentToken, pricePerImage, imageDescription,
    refundTimeLimit);

    emit AgentConfigured(agentToken, pricePerImage, imageDescription,
    refundTimeLimit);
}
```

Spectral: Resolved with @676e89718f6...

Zenith: Verified.

[M-5] The owner cannot set a new creator for the agent tokens

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

Target

- AgentImageService.sol

Description:

When the `owner` calls the `updateAgentCreator` function to set a new `creator`, it internally calls the `updateAgentCreator` function of the `deployer`.

- AgentImageService.sol#L354

```
function updateAgentCreator(
    address agentToken,
    address newCreator
) external onlyOwner {
    require(agentToken ≠ address(0), "Agent token cannot be zero address");
    require(newCreator ≠ address(0), "New creator cannot be zero address");
    deployer.updateAgentCreator(agentToken, newCreator);
}
```

However, the `deployer`'s `updateAgentCreator` function can only be called by the current `creator` of that `agent token`—not by `AgentImageService`.

- AutonomousAgentDeployer.sol#L255-L259

```
function updateAgentCreator(address _agentToken, address newCreator)
    external onlyAgentCreator(_agentToken) whenNotPaused {
    require(newCreator ≠ address(0), "ZERO_ADDRESS");
    distributor.setAgentCreator(_agentToken, newCreator);
    emit AgentCreatorUpdated(_agentToken, newCreator);
}
```

As a result, the transaction will be reverted.

Recommendations:

Grant the `AgentImageService` permission to call the `deployer`'s `updateAgentCreator` function.

Spectral: Acknowledged. This function will only be used once when the AgentImageService is the first owner of the agent before transferring it to the agent creator. It is not intended to be reused after deployment.

4.3 Low Risk

A total of 2 low risk findings were identified.

[L-1] setDistributor and setSpectralStaking functions might be needed

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

Target

- AutonomousAgentDeployer.sol

Description:

v2 removes `setDistributor` and `setSpectralStaking` entirely, so if a bad address passed into `initialize(...)`, it can't be corrected later.

Recommendations:

Recommended keeping the functions.

Spectral: Acknowledged. Due to bytecode size, we removed them and will add them with an upgrade at anytime we need to change the variables which might never happen.

[L-2] The fees can be transferred to the admin

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

Target

- AgentImageService.sol

Description:

The `withdrawFees` function can be called by either the `creator` of the `agent tokens` or the `admin`.

- AgentImageService.sol#L311

```
function withdrawFees(address agentToken)
    external onlyAgentOwner(agentToken) nonReentrant {
    uint256 amount = accumulatedFees[agentToken];
    require(amount > 0, "No fees to withdraw");

    accumulatedFees[agentToken] = 0;
    IERC20Upgradeable(agentToken).safeTransfer(msg.sender, amount);

    emit FeesWithdrawn(agentToken, amount);
}
```

The `fees` are then sent to the `caller` (`msg.sender`). This means the `admin` can collect `fees` for any `agent token`.

Recommendations:

```
function withdrawFees(address agentToken)
    external onlyAgentOwner(agentToken) nonReentrant {
    uint256 amount = accumulatedFees[agentToken];
    require(amount > 0, "No fees to withdraw");

    accumulatedFees[agentToken] = 0;
    IERC20Upgradeable(agentToken).safeTransfer(msg.sender, amount);
```

```
        IERC20Upgradeable(agentToken).safeTransfer(IOctoDistributor(address(
            deployer.distributor())).agentCreators(agentToken), amount);

        emit FeesWithdrawn(agentToken, amount);
    }
```

Spectral: Resolved with @42034b4efaa...

Zenith: Verified.

4.4 Informational

A total of 5 informational findings were identified.

[I-1] Unused mapping, variable and events in V2

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

Target

- AutonomousAgentDeployer.sol

Description:

In AutonomousAgentDeployer V2 below items are not used; • `mapping(address⇒address) public agentWallets;` • `uint256 accumulatedTaxToSwap;` • `event DistributorSet` and `event SpectralStakingSet` are declared but never emitted (since the setters that used them were removed).

Recommendations:

Remove the unused events, variable and mapping

Spectral: Resolved with @cd55ade7f2f...

Zenith: Verified.

[I-2] Recommendation for public mempool chain deployments

SEVERITY: Informational	IMPACT: Informational
STATUS: Acknowledged	LIKELIHOOD: High

Target

- AgentImageService.sol

Description:

If this contract is ever intended to be deployed on the chains with public mempool, the `deployImageAgent` function is subject to be MEV'ed and possibly poison the approvals of the `agentToken` by adversaries.

```
function deployImageAgent(
        string memory agentName,
        string memory agentTicker,
        string memory agentDescription,
        address owner,
        uint256 minSpecAmount) external payable returns (address agentToken)
    {
        require(owner ≠ address(0), "AgentImageService: Owner cannot be
    zero address");
        require(msg.value >=
    deployer.parameters(IAutonomousAgentDeployer.Parameter.DEPLOYMENT_COST_ETH),
    "AgentImageService: Incorrect ETH amount sent for deployment");
        // only transfer ETH to the this contract from the user in the next
    lines
        (agentToken) = deployer.deployAgentWithETH{value:
    msg.value}(agentName, agentTicker, minSpecAmount);
        agents[agentToken] = Agent({
            agentName: agentName,
            agentTicker: agentTicker,
            agentDescription: agentDescription
        });
        agentConfigs[agentToken] = AgentConfig({
            pricePerImage: 1 ether,
            imageDescription: "",
            refundTimeLimit: 1 days,
            isConfigured: true
        });
        uint256 tokenAmountOut
    = IERC20Upgradeable(agentToken).balanceOf(address(this));
        IERC20Upgradeable(agentToken).safeTransfer(owner, tokenAmountOut);
        deployer.updateAgentCreator(agentToken, owner);
        emit ImageAgentDeployed(agentName, agentTicker, agentDescription,
    owner, agentToken, minSpecAmount, tokenAmountOut);
    }
```

Recommendations:

Include `msg.sender` in `agentToken` salt params.

Spectral: Acknowledged.

[I-3] Unused variables

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

Target

- AgentImageService.sol
- AgentImageService.sol

Description:

The contract declares two mappings intended for tracking all requests and their processed state:

```
mapping(address ⇒ bytes32[]) private agentRequestIds;
mapping(address ⇒ mapping(bytes32 ⇒ bool)) private requestProcessed;
```

However, no function ever writes to `agentRequestIds` or sets entries in `requestProcessed`. Every time a user calls `requestImage()`, the new request is stored in `imageRequests`, but it is never appended to `agentRequestIds`. Likewise, when a request is fulfilled or refunded, `requestProcessed` is never updated.

Recommendations:

Remove these two unused mappings.

Spectral: Resolved with @d92ecc4740...

Zenith: Verified.

[I-4] The AgentImageService should inherit from IAgentImageService

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

Target

- AgentImageService.sol

Description:

The `IAgentImageService` contains a duplicated `AgentConfig` struct and an incorrect `ImageRequest` struct.

- IAgentImageService.sol#L5-L12

```
interface IAgentImageService {
    struct ImageRequest {
        address user;
        uint256 amount;
        uint256 timestamp;
        bool fulfilled;
        bool refunded;
        string prompt;
    }

    struct AgentConfig {
        uint256 pricePerImage;
        string imageDescription;
        uint256 refundTimeLimit;
        bool isConfigured;
    }
}
```

Recommendations:

Move the structs and events to `IAgentImageService` and inherit from it. Alternatively, remove these structs from `IAgentImageService` altogether.

Spectral: Resolved with @0230822d573....

Zenith: Verified.

[I-5] The previously calculated value can be reused

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

Target

- AgentImageService.sol

Description:

In the `fulfillImage` function, `treasuryCut` can be used on the last line.

- AgentImageService.sol#L245

```
function fulfillImage(
    address agentToken,
    bytes32 requestId
) external onlyAgentOwner(agentToken) onlyValidAgent(agentToken)
    nonReentrant {
    pendingFees[agentToken] -= request.amount;
    uint256 treasuryCut = (request.amount
    * parameters[Parameter.TREASURY_CUT]) / 10000;
    accumulatedFees[agentToken] += (request.amount - treasuryCut);
    IERC20Upgradeable(agentToken).safeTransfer(spectralTreasury,
    (request.amount * parameters[Parameter.TREASURY_CUT]) / 10000);
}
```

Recommendations:

```
function fulfillImage(
    address agentToken,
    bytes32 requestId
) external onlyAgentOwner(agentToken) onlyValidAgent(agentToken)
    nonReentrant {
    pendingFees[agentToken] -= request.amount;
    uint256 treasuryCut = (request.amount
    * parameters[Parameter.TREASURY_CUT]) / 10000;
    accumulatedFees[agentToken] += (request.amount - treasuryCut);
    IERC20Upgradeable(agentToken).safeTransfer
  (spectralTreasury,
  (request.amount * parameters[Parameter.TREASURY_CUT]) / 10000);
    IERC20Upgradeable(agentToken).safeTransfer(spectralTreasury,
 treasuryCut);
 }
```

Spectral: Resolved with @4b6c6e4825c....

Zenith: Verified.