SALUS SECURITY

MAR 2024

CODE SECURITY ASSESSMENT

ORA

Overview

Project Summary

- Name: ORA - OAO
- Platform: Ethereum
- Language: Solidity
- Repository:
    - https://github.com/ora-io/OAO-Audit
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	ORA - OAO
Version	v1
Type	Solidity
Dates	Mar 20 2024
Logs	Mar 20 2024

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	2
Total Low-Severity issues	5
Total informational issues	5
Total	14

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Unable to start a challenge because submitBlockTime initialization is missing	High	Business Logic	Pending
2	The challenge process is incomplete	High	Business Logic	Pending
3	Users' requests may fail due to inconsistent model data between AIOracle and DummyOmpl	Medium	Business Logic	Pending
4	Centralization risk	Medium	Centralization	Pending
5	The first user to request a callback only pays for the model	Low	Business Logic	Pending
6	Use transfer() for native token transfer	Low	Business Logic	Pending
7	The user's callback function may not be called in time	Low	Business Logic	Pending
8	Revenue from removed models may be withdrawn by the owner	Low	Business Logic	Pending
9	Use of the testnet address	Low	Configuration	Pending
10	Lack of validation of request output	Informational	Data Validation	Pending
11	Improper error message in AIOracle::_validateParams()	Informational	Logging	Pending
12	No functionality to remove whitelist addresses	Informational	Business Logic	Pending
13	Spelling mistakes	Informational	Code Quality	Pending
14	Brute force CREATE2 salt in the contract	Informational	Business Logic	Pending

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Unable to start a challenge because submitBlockTime initialization is missing

Severity: High	Category: Business Logic
Target: - contracts/DummyOpml.sol	

Description

Submitters will upload AI results according to users' requests. Validators can verify the result and start a challenge if uploaded results are incorrect. Validators are supposed to start a challenge within the challenge period starting from the submitBlockTime.

contracts/DummyOpml.sol:L189-L193

```
function isFinalized(uint256 requestId) public view ifRequestExists(requestId) returns
(bool) {
        RequestData storage request = requests[requestId];
        return block.number - request.submitBlockTime >= challengePeriod;
        // return request.isFinalized;
}
```

The submitBlockTime is expected to be set when submitters upload AI results. However, this step is missing in the current implementation.

contracts/DummyOpml.sol:L129-L134

```
function uploadResult(uint256 requestId, bytes calldata output) public onlyWhitelisted
ifRequestExists(requestId) {
    RequestData storage request = requests[requestId];
    require(request.output.length == 0, "to upload request, make sure that the request
is never served by others. If you think the provided result is incorrect, please
challenge it!");
    request.output = output;
    request.submitter = msg.sender;
}
```

Recommendation

Consider updating requests' submitBlockTime when uploadResult() is called.

2. The challenge process is incomplete

Severity: High	Category: Business Logic

Target:
- contracts/DummyOpml.sol

Description

Validators will check uploaded results by submitters and can start a challenge if they find the provided result is incorrect. Some processes are needed to check whether the challenge is valid. If a challenge is valid, the previously uploaded result needs to be updated. In the current implementation, validators can start a challenge, but the subsequent steps are missing.

contracts/DummyOpml.sol:L138-L175

```
function startChallenge(
    uint256 requestId,
    bytes calldata output,
    bytes32 finalState,
    uint256 stepCount
) external onlyWhitelisted ifRequestExists(requestId) returns (uint256) {
    RequestData storage request = requests[requestId];
    ...
    // TODO: check the outputHash is consistent with the finalState!
    request.isInChallenge = true;
    // Write input hash at predefined memory address.
    bytes32 startState = request.startState;

    uint256 challengeId = lastChallengeId++;
    ChallengeData storage c = challenges[challengeId];
    request.challengeId = challengeId;
    c.requestId = requestId;

    // A NEW CHALLENGER APPEARS
    c.outputHash = keccak256(output);
    c.challenger = payable(msg.sender);
    c.submitter = payable(request.submitter);
    ...

    return challengeId;
}
```

Recommendation

Consider completing the challenge process.

3. Users' requests may fail due to inconsistent model data between AIOracle and DummyOmpl

Severity: Medium	Category: Business Logic

Target:
- contracts/DummyOpml.sol
- contracts/AIOracle.sol

Description

In the AIOracle contract, the owner can add new models via the uploadModel() function, and the model's hash data will be updated to the DummyOpml contract at the same time.

contracts/AIOracle.sol:L153-L165

```
function uploadModel(uint256 modelId, bytes32 modelHash, bytes32 programHash, uint256
fee, address receiver, uint256 receiverPercentage) external onlyOwner {
        ...
        ModelData storage model = models[modelId];
        model.modelHash = modelHash;
        model.programHash = programHash;
        ...
        opml().uploadModel(modelHash, programHash);
}
```

contracts/DummyOpml.sol:L103-L106

```
function uploadModel(bytes32 modelHash, bytes32 programHash) external onlyWhitelisted {
        // require(!modelExists[modelHash][programHash], "the model already exists");
        modelExists[modelHash][programHash] = true;
}
```

When a user sends a request, AIOracle will pass the related model's hash data to the DummyOpml contract, which will validate the model and save the request.

contracts/DummyOpml.sol:L117

```
function initOpmlRequest(bytes32 modelHash, bytes32 programHash, bytes calldata input)
external onlyWhitelisted returns (uint256 requestId) {
    require(modelExists[modelHash][programHash], "model does not exist");
    ...
}
```

Meanwhile, in the AIOracle contract, the model's data can be updated via the updateModel() function but updated hash data will not be synchronized to the DummyOpml contract. Once the modelHash or programHash of a model changes, users' requests for the model will fail due to the model existence check in the DummyOpml contract.

contracts/AIOracle.sol:L167-L175

```
function updateModel(uint256 modelId, bytes32 modelHash, bytes32 programHash, uint256
fee, address receiver, uint256 receiverPercentage) external onlyOwner
ifModelExists(modelId) {
    require(receiverPercentage <= 100, "percentage should be <= 100");
    ModelData storage model = models[modelId];
    model.modelHash = modelHash;
```

```
    model.programHash = programHash;
    model.fee = fee;
    model.receiver = receiver;
    model.receiverPercentage = receiverPercentage;
}
```

Recommendation

Consider updating the modelExists variable in the DummyOpml contract when updating models in the AIOracle contract.

4. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/AIOracle.sol	

Description

There is a privileged owner role in the AIOracle contract. The owner of the AIOracle contract can manage the blacklist, set model parameters, and withdraw revenue.

Should the owner's private key be compromised, an attacker could remove all models and withdraw revenue.

Since the privileged account is a plain EOA account, this can be worrisome and pose a risk to the other users.

contracts/AIOracle.sol:L14-L16

```
function owner() public pure returns (address) {
        return 0xf5aeB5A4B35be7Af7dBfDb765F99bCF479c917BD;
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

5. The first user to request a callback only pays for the model

Severity: Low	Category: Business Logic
Target: - contracts/AIOracle.sol	

Description

When a user requests a callback, sufficient ether must be transferred to the oracle to cover model fees and gas fees.

contracts/AIOracle.sol:L189

```
require(msg.value >= model.fee + gasPrice * gasLimit, "insufficient fee");
```

However, the initial gas price is 0 and is set when the first callback is invoked. Thus, the first user to request a callback only pays for the model.

contracts/AIOracle.sol:L287

```
function invokeCallback(uint256 requestId, bytes calldata output) external onlyServer {
        ...
        emit AICallbackResult(msg.sender, requestId, output);
        gasPrice = tx.gasprice;
}
```

Recommendation

Consider setting an appropriate initial value for the gasPrice variable.

6. Use transfer() for native token transfer

Severity: Low	Category: Business Logic
Target: - contracts/AIOracle.sol	

Description

The withdraw() and claimModelRevenue() functions in the AIOracle contract use the transfer() function to transfer native tokens to the receiver. The transfer() function is not recommended for sending native tokens due to its 2300 gas unit limit which may not work with smart contract wallets or multi-sig.

contracts/AIOracle.sol:L228-L233

```
function claimModelRevenue(uint256 modelId) external ifModelExists(modelId) {
    ModelData storage model = models[modelId];
    require(model.accumulateRevenue > 0, "accumulate revenue is 0");
    payable(model.receiver).transfer(model.accumulateRevenue);
    model.accumulateRevenue = 0;
}
```

Recommendation

Consider using call() with value instead of transfer().

In addition, following the CEI pattern, update model.accumulateRevenue before sending native tokens to avoid reentrancy attacks when using call().

7. The user's callback function may not be called in time

Severity: Low	Category: Business Logic

Target:
- contracts/DummyOpml.sol
- contracts/AIOracle.sol

Description

After users submit a request, results can be uploaded in two ways: either by submitters through the DummyOpml::uploadResult() method, or by the server via AIOracle::invokeCallback(). If results are uploaded by submitters, no callback function will be invoked. Meanwhile, calls from the server will also be blocked due to the request output length check. In this case, the callback can only be triggered via AIOracle::updateResult() and the gas fee needs to be paid again.

contracts/AIOracle.sol:L264-L281

```
function invokeCallback(uint256 requestId, bytes calldata output) external onlyServer {
    // read request of requestId
    AICallbackRequestData storage request = requests[requestId];

    // others can challenge if the result is incorrect!
    opml().uploadResult(requestId, output);

    // invoke callback
    if(request.callbackContract != address(0)) {
        bytes memory payload = abi.encodeWithSelector(callbackFunctionSelector(),
request.requestId, output, request.callbackData);
        (bool success, bytes memory data) = request.callbackContract.call{gas:
request.gasLimit}(payload);
    ...
    }
    ...
}
```

contracts/DummyOpml.sol:L129-L134

```
function uploadResult(uint256 requestId, bytes calldata output) public onlyWhitelisted
ifRequestExists(requestId) {
    RequestData storage request = requests[requestId];
    require(request.output.length == 0, "to upload request, make sure that the request
is never served by others. If you think the provided result is incorrect, please
challenge it!");
    request.output = output;
    request.submitter = msg.sender;
}
```

Recommendation

Consider limiting the allowed callers of the uploadResult() function to the AIOracle contract.

8. Revenue from removed models may be withdrawn by the owner

Severity: Low	Category: Business Logic
Target: - contracts/AIOracle.sol	

Description

The owner can withdraw all ethers in the AIOracle contract except for the accumulated revenue of all active models stored in modelIDs.

contracts/AIOracle.sol:L116-L125

```
function withdraw() external onlyOwner {
    uint256 modelRevenue;
    for (uint i = 0; i < modelIDs.length; i++) {
        uint256 id = modelIDs[i];
        ModelData memory model = models[id];
        modelRevenue += model.accumulateRevenue;
    }
    uint256 ownerRevenue = address(this).balance - modelRevenue;
    payable(msg.sender).transfer(ownerRevenue);
}
```

Removing a model does not transfer the accumulated revenue to the model receiver, nor does it clear the record of the accumulated revenue. The owner can withdraw revenue from removed models. Since the accumulated revenue record is not reset and can be inherited by new models with the same model ID, their receivers may claim other models' revenue.

contracts/AIOracle.sol:L92-L94

```
function resetModelIDs() external onlyOwner {
        delete modelIDs;
}
```

contracts/AIOracle.sol:L101-L114

```
function removeModel(uint256 modelId) external onlyOwner ifModelExists(modelId) {
        modelExists[modelId] = false;
        // remove from modelIDs
        for (uint i = 0; i < modelIDs.length; i++) {
            uint256 id = modelIDs[i];
            if (id == modelId) {
                // Replace the element at index with the last element
                modelIDs[i] = modelIDs[modelIDs.length - 1];
                // Remove the last element by reducing the array's length
                modelIDs.pop();
                break;
            }
        }
}
```

Recommendation

Please revisit the logic related to model ID. Consider sending the accumulated revenue to the model receiver and resetting the record before removing the model.

9. Use of the testnet address

Severity: Low	Category: Configuration
Target: - contracts/AIOracle.sol	

Description

The OPML has been set to a testnet address in the AIOracle contract. This address will be used when interacting with the OPML.

contracts/AIOracle.sol:L10-L12

```
function opml() public pure returns (IOpml) {
        return IOpml(0xa8a416519CD5dd60c1Dc97B195443D3263a6cb60);
}
```

Recommendation

Consider configuring the correct address for OPML before deploying to the mainnet.

2.3 Informational Findings

10. Lack of validation of request output

Severity: Informational	Category: Data Validation
Target: - contracts/DummyOpml.sol	

Description

There is no zero-length validation for output provided by whitelisted accounts. Since the uploadResult() function uses the length of the request output to determine whether upload can be performed, if the output length provided by whitelisted accounts is 0, the uploadResult() function can be called again.

Recommendation

Consider adding a zero-length check for output before assigning it to the request output.

11. Improper error message in AIOracle::_validateParams()

Severity: Informational	Category: Logging
Target: - contracts/AIOracle.sol	

Description

In AIOracle::_validateParams(), the system will validate all parameters using in the requestCallback() function. If there is no callback, gasLimit should be zero. If there is a callback, gasLimit should not be zero. The error message “gasLimit cannot be 0” is not descriptive enough.

contracts/AIOracle.sol:L192-L193

```
bool noCallback = callbackContract == address(0);
require(noCallback == (gasLimit == 0), "gasLimit cannot be 0");
```

Recommendation

Consider changing the error message to "Invalid gasLimit".

12. No functionality to remove whitelist addresses

Severity: Informational	Category: Business Logic
Target: - contracts/DummyOpml.sol	

Description

The DummyOpml contract allows accounts in the whitelist to upload models, init requests, upload results, etc. However, there is no functionality to remove addresses from the whitelist. This could be problematic if a whitelist address is discovered to be malicious.

Recommendation

Consider adding a privileged function to remove addresses from the whitelist.

13. Spelling mistakes

Severity: Informational	Category: Code Quality

Target:
- contracts/DummyOpml.sol
- contracts/Prompt.sol

Description

contracts/DummyOpml.sol:L38

```
// RequstId
```

"RequstId" should be "Requ**e**stId".

contracts/DummyOpml.sol:L147

```
require(!request.isInChallenge, "the requst is still in challenge, please wait for the
end of the challenge");
```

"requst" should be "requ**e**st".

contracts/Prompt.sol:L8

```
// this contract is for ai.hyperoracle.io websie
```

"websie" should be "websi**t**e".

Recommendation

Consider correcting the spelling mistakes.

14. Brute force CREATE2 salt in the contract

Severity: Informational	Category: Business Logic
Target: - contracts/upgrade/ProxyFactory.sol	

Description

The deployBrute() function in the ProxyFactory contract is utilized to find a salt value used in contract creation through CREATE2, ensuring that the deployed contract address starts and ends with 0a0. It can only try 200 salts at a time, making the search less efficient.

contracts/upgrade/ProxyFactory.sol:L56-L89

```
function deployBrute(address impl, uint256 tick) public view returns(uint256) {
        ...
        for (uint i = tick; i < tick + 200; i++) {
            bytes32 salt = bytes32(i);
            ...
            address predictAddress = address(
               uint160(
               uint(
                       keccak256(
                       abi.encodePacked(
                               bytes1(0xFF),
                               address(this),
                               salt,
                               bytecodeHash
                       )
                       )
               )
               )
            );

            if(_doesAddressStartWith(predictAddress, 0x0a0)) {
               ...
               return i;
            }
        }
        return 0;
}
```

Recommendation

It is recommended to use cast create2 to find the salt more efficiently.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 65d0215:

File	SHA-1 hash
contracts/AIOracleCallbackReceiver.sol	d659c8b8d02810ecf6c32f978582ebc558dfc7e2
contracts/AIOracle.sol	c341b773a8c298019ef6c358cb28819d2e2d2162
contracts/DummyOpml.sol	159be26f55e8ad467c1f45e78d9163eb4131138b
contracts/Prompt.sol	d489f85457c97842b93ad6a1bcbec48e960d870b
contracts/upgrade/ProxyFactory.sol	b296d3f6d073c38feedd162c7e3f9e76c10b25bd