// HALBORN

ORAICHAIN
Orai Oracle
Smart Contract Security Audit

Prepared by: **Halborn**
Date of Engagement: December 14, 2020
Visit: Halborn.com

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	12/14/2020	Gabi Urrutia
0.2	Document Edits	12/17/2020	Gabi Urrutia
1.0	Final Version	12/21/2020	Gabi Urrutia

CONTACT	COMPANY	EMAIL
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

1.1 INTRODUCTION

Oraichain engaged Halborn to conduct a security assessment on their Oracle smart contract beginning on December 14th, 2020 and ending December 21th, 2020. The security assessment was scoped to the contract orai-oracle.sol and an audit of the security risk and implications regarding the changes introduced by the development team at Oraichain prior to its production release shortly following the assessments deadline.

The Oracle smart contract does not import any external libraries but, the contract orai-oracle.sol is made up of 7 contracts: OraiTokenReceiver, OracleRequestInterface, OracleInterface, OraiTokenInterface, Ownable, SafeMath and Oracle. Therefore, the contract works by itself without importing any external contracts, increasing its security.

Overall, the smart contracts code is extremely well documented, follows a high-quality software development standard, contain many utilities and automation scripts to support continuous deployment / testing / integration, and does NOT contain any obvious exploitation vectors that Halborn was able to leverage within the timeframe of testing allotted.

Though the outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties related to the Oracle Contract was performed to achieve objectives and deliverables set in the scope. It is important to remark the use of the best practices for secure smart contract development.

Halborn recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts. External threats, such as economic attacks, oracle attacks, and inter-contract functions and calls should be validated for expected logic and state.

1.2 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture, purpose, and use of Oracle.
- Smart Contract manual code read and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual Assessment of use and safety for the critical solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Scanning of solidity files for vulnerabilities, security hotspots, or bugs. (MythX)
- Static Analysis of security for scoped contract and imported functions. (Slither)
- Smart Contract analysis and automatic exploitation (limited-time)
- Symbolic Execution / EVM bytecode security assessment (limited-time)

1.3 SCOPE

IN-SCOPE:
Code related to the Oracle smart contract. Specific commit of contract: commit a96f559dce4f5ce9673ebb31ca710db499d38453

OUT-OF-SCOPE:
Other smart contracts in the repository and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	0

SECURITY ANALYSIS	RISK LEVEL
USE OF INLINE ASSEMBLY	Informational
POSSIBLE MISUSE OF PUBLIC FUNCTIONS	Informational
STATIC ANALYSIS REPORT	Informational
AUTOMATED SECURITY SCAN	Informational

FINDINGS & TECH DETAILS

3.1 USE OF INLINE ASSEMBLY - INFORMATIONAL

Description:

Inline assembly is a way to access the Etehreum Virtual Machine at a low level. This discards several important safety features in Solidity.

Code Location:

Orai-oracle.sol Line #19-22

```
19          assembly {
20              mstore(add(_data, 36), _sender) // ensure correct sender is passed
21              mstore(add(_data, 68), _amount)    // ensure correct amount is passed
22          }
```

Recommendation:

When possible, do not use inline assembly because it is a manner to access to the EVM (Ethereum Virtual Machine) at a low level. An attacker could bypass many important safety features of Solidity.

3.2 POSSIBLE MISUSE OF PUBLIC FUNCTIONS - INFORMATIONAL

Description:

In public functions, array arguments are immediately copied array to memory, while external functions can read directly from calldata. Reading calldata is cheaper than memory allocation. Public functions need to write the arguments to memory because public functions may be called internally. Internal calls are passed internally by pointers to memory. Thus, function expects its arguments being in memory when the compiler generates the code for an internal function. In orai-oracle contract, onTokenTransfer and transferOwnership functions are never directly called by another function in the same contract.

Code Location:

orai-oracle.sol Line #10-25

```
10      function onTokenTransfer(
11          address _sender,
12          uint256 _amount,
13          bytes memory _data
14      )
15      public
16      onlyOrai
17      validRequestLength(_data)
18      {
19          assembly {
20              mstore(add(_data, 36), _sender) // ensure correct sender is passed
21              mstore(add(_data, 68), _amount)    // ensure correct amount is passed
22          }
23          (bool success,) = address(this).delegatecall(_data);
24          require(success, "Unable to create request");
25      }
```

orai-oracle.sol Line #135-137

```
135     function transferOwnership(address newOwner) public onlyOwner {
136         _transferOwnership(newOwner);
137     }
```

Recommendation:

Consider as much as possible declaring external variables instead of public variables. As for best practices, you should use external if you expect that the function will only ever be called externally and use public if you need to call the function internally. In that case, both functions are not called by another function in the same contract, so marking both function as external can save gas.

3.3 STATIC ANALYSIS REPORT - INFORMATIONAL

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the Oracle contract. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire codebase.

Results:

```
INFO:Detectors:
OraiTokenReceiver.onTokenTransfer(address,uint256,bytes) (orai-oracle.sol#10-25) uses assembly
        - INLINE ASM (orai-oracle.sol#19-22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Low level call in OraiTokenReceiver.onTokenTransfer(address,uint256,bytes) (orai-oracle.sol#10-25):
        - (success) = address(this).delegatecall(_data) (orai-oracle.sol#23)
Low level call in Oracle.fulfillOracleRequest(bytes32,address,bytes4,bytes) (orai-oracle.sol#279-303):
        - (success) = _callbackAddress.call(abi.encodeWithSelector(_callbackFunctionId,_requestId,_data)) (orai-oracle.sol#300)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter OraiTokenReceiver.onTokenTransfer(address,uint256,bytes)._data (orai-oracle.sol#13) is not in mixedCase
Parameter Oracle.oracleRequest(address,uint256,bytes32,address,bytes4,uint256,bytes)._sender (orai-oracle.sol#238) is not in mixedCase
Parameter Oracle.oracleRequest(address,uint256,bytes32,address,bytes4,uint256,bytes)._payment (orai-oracle.sol#239) is not in mixedCase
Parameter Oracle.oracleRequest(address,uint256,bytes32,address,bytes4,uint256,bytes)._specId (orai-oracle.sol#240) is not in mixedCase
Parameter Oracle.oracleRequest(address,uint256,bytes32,address,bytes4,uint256,bytes)._callbackAddress (orai-oracle.sol#241) is not in mixedCase
Parameter Oracle.oracleRequest(address,uint256,bytes32,address,bytes4,uint256,bytes)._callbackFunction (orai-oracle.sol#242) is not in mixedCase
Parameter Oracle.oracleRequest(address,uint256,bytes32,address,bytes4,uint256,bytes)._nonce (orai-oracle.sol#243) is not in mixedCase
Parameter Oracle.fulfillOracleRequest(bytes32,address,bytes4,bytes)._requestId (orai-oracle.sol#280) is not in mixedCase
Parameter Oracle.fulfillOracleRequest(bytes32,address,bytes4,bytes)._callbackAddress (orai-oracle.sol#281) is not in mixedCase
Parameter Oracle.fulfillOracleRequest(bytes32,address,bytes4,bytes)._callbackFunctionId (orai-oracle.sol#282) is not in mixedCase
Parameter Oracle.fulfillOracleRequest(bytes32,address,bytes4,bytes)._data (orai-oracle.sol#283) is not in mixedCase
Parameter Oracle.getAuthorizationStatus(address)._node (orai-oracle.sol#306) is not in mixedCase
Parameter Oracle.setFulfillmentPermission(address,bool)._node (orai-oracle.sol#321) is not in mixedCase
Parameter Oracle.setFulfillmentPermission(address,bool)._allowed (orai-oracle.sol#321) is not in mixedCase
Parameter Oracle.withdraw(uint256)._amount (orai-oracle.sol#334) is not in mixedCase
Variable Oracle.OraiToken (orai-oracle.sol#202) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
Oracle.slitherConstructorConstantVariables() (orai-oracle.sol#195-383) uses literals with too many digits:
        - MINIMUM_CONSUMER_GAS_LIMIT = 400000 (orai-oracle.sol#198)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
onTokenTransfer(address,uint256,bytes) should be declared external:
        - OraiTokenReceiver.onTokenTransfer(address,uint256,bytes) (orai-oracle.sol#10-25)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (orai-oracle.sol#135-137)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

3.4 AUTOMATED SECURITY SCAN - INFORMATIONAL

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruit on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the testers machine and sent the compiled results to the analyzers to locate any vulnerabilities. Security Detections are only in scope, and the analysis was pointed towards issues with orai-oracle.

Results

aiVaultORAI

MythX detected 0 High findings, 2 Medium, and 0 Low.

DETECTED ISSUES

0 High | 2 Medium | 0 Low

ID	SEVERITY	NAME	FILE	LOCATION
SWC-000	Medium	Function could be marked as external.	orai-oracle.sol	L: 10 C: 4
SWC-000	Medium	Function could be marked as external.	orai-oracle.sol	L: 135 C: 4

THANK YOU FOR CHOOSING

// HALBORN