DEOR - ERC20 & Oracles

Security Code Review

https://twitter.com/VidarTheAuditor - 9 March 2021

Overview

Project Summary

Project Name	DEOR
Description	Decentrlized Oracle
Platform	**Ethereum**, Solidity, JavaScript/Node
Contracts	https://github.com/computercybersecurity/DecentralizedOracle commit 051ac599074da031458ea0b04ee3ae68707a6738
	• **DEOR - 0x63726dAe7C57d25e90ec829ce9a5C745Ffd984d3**

Executive Summary

Solidity contracts (as On-Chain part) and JS/Node source files (Off-Chain part) have been provided as a decentralised oracle solution.

We have run extensive static analysis of the codebase as well as standard security assessment utilising industry approved tools. We have also tested the system against the white paper description.

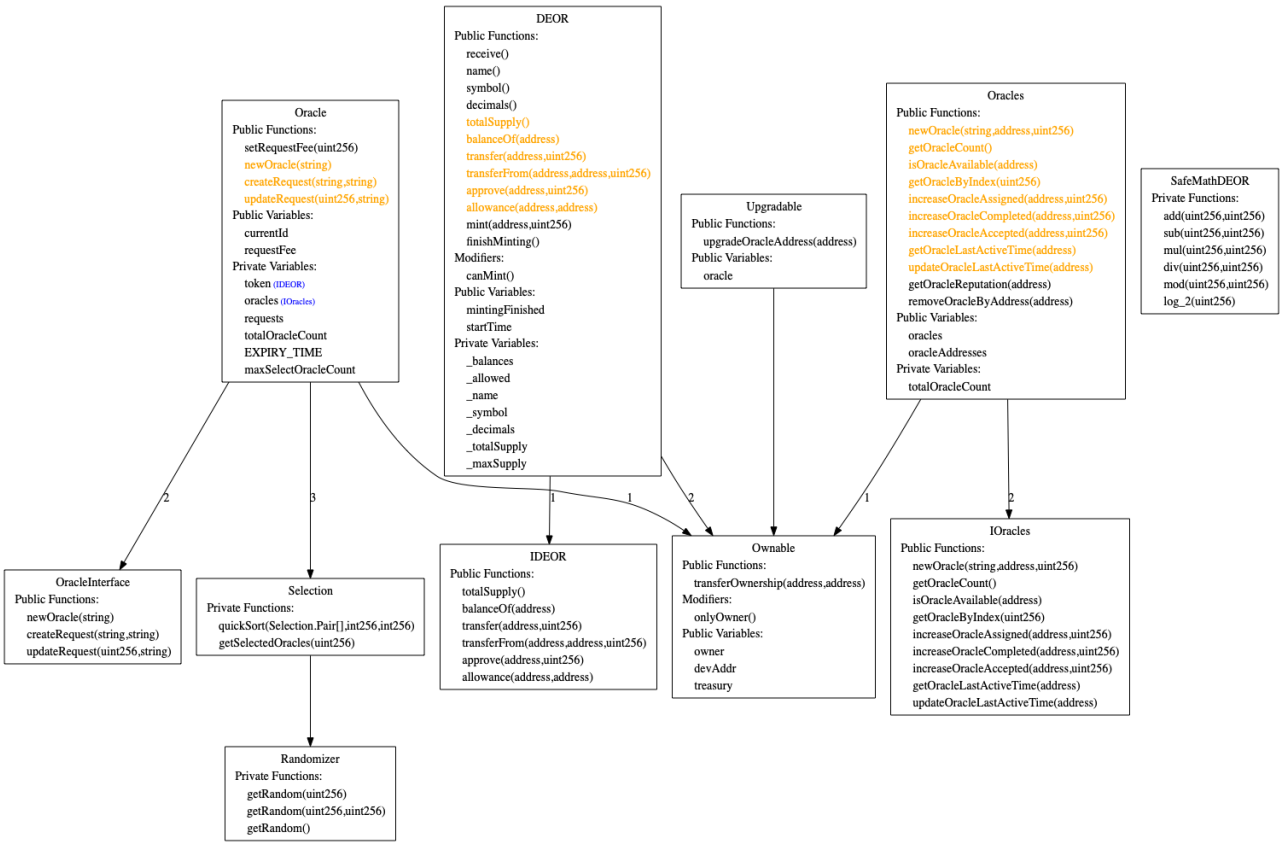
We have not found any critical vulnerabilities arising from a third party involvement.

Some recommendations where issued in Deployment section.

Disclaimer: The analysis did not include any tokenomics analysis (e.g. APY rates etc).

Architecture & Standards

Please find below the calling architecture of the reviewed contracts.



DEOR is fully ERC-20 compatible.

```
# Check DEOR

## Check functions
[✓] totalSupply() is present
        [✓] totalSupply() -> () (correct return value)
        [✓] totalSupply() is view
[✓] balanceOf(address) is present
        [✓] balanceOf(address) -> () (correct return value)
        [✓] balanceOf(address) is view
[✓] transfer(address,uint256) is present
        [✓] transfer(address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] transferFrom(address,address,uint256) is present
        [✓] transferFrom(address,address,uint256) -> () (correct return value)
        [✓] Transfer(address,address,uint256) is emitted
[✓] approve(address,uint256) is present
        [✓] approve(address,uint256) -> () (correct return value)
        [✓] Approval(address,address,uint256) is emitted
[✓] allowance(address,address) is present
        [✓] allowance(address,address) -> () (correct return value)
        [✓] allowance(address,address) is view
[✓] name() is present
        [✓] name() -> () (correct return value)
        [✓] name() is view
[✓] symbol() is present
        [✓] symbol() -> () (correct return value)
        [✓] symbol() is view
[✓] decimals() is present
        [ ] decimals() -> () should return uint8
        [✓] decimals() is view

## Check events
[✓] Transfer(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
[✓] Approval(address,address,uint256) is present
        [✓] parameter 0 is indexed
        [✓] parameter 1 is indexed
```

Findings

Number of contracts: 11 (including inherited ones)

Use: SafeMath

Name	# functions	ERCS	ERC20 info	Complex code	Features
DEOR	22	ERC20	Minting Approve Race Cond.	No	Receive ETH
Oracle	17			Yes	Tokens interaction
Oracles	24			No	
Upgradable	5			No	Upgradeable
SafeMathDEOR	6			No	

The minting on DEOR was part of the functionality of the system. The total supply of the token has been already minted. The minting has been finished and it is not possible to mint more tokens.

For more info see Deployment section.

```
ftrace | funcSig
78  modifier canMint() {
79      require(!mintingFinished);
80      _;
81  }
82
83  /**
84  * Function to mint tokens
85  * @param _to The address that will recieve the minted tokens.
86  * @param _amount The amount of tokens to mint.
87  * @return A boolean that indicates if the operation was successful.
88  */
89  function mint(address _to, uint256 _amount) public onlyOwner canMint returns (bool) {
90      uint256 amount = _maxSupply.sub(_totalSupply);
91      if (amount > _amount) {
92          amount = _amount;
93      }
        ftrace | funcSig
94      else {
95          mintingFinished = true;
96          emit MintFinished();
97      }
98      _totalSupply = _totalSupply.add(amount);
99      _balances[_to] = _balances[_to].add(amount);
100     emit Mint(_to, _amount);
101     return true;
102 }
103
104 /**
105 * Function to stop minting new tokens.
106 * @return True if the operation was successful.
107 */
108 function finishMinting() public onlyOwner returns (bool) {
109     mintingFinished = true;
110     emit MintFinished();
111     return true;
112 }
```

Static Analysis Findings

High issues: None

Medium issues:

Unused Return:

```
Oracle.updateRequest(uint256,string) (contracts/Oracle.sol#87-154) ignores return value by token.transferFrom(owner,addr,awardForRequest + penaltyForReq
uest) (contracts/Oracle.sol#140)
```

The return value of an external call is not stored in a local or state variable.

[Recommendation] Ensure that all the return values of the function calls are used..

Possible Reentrancy:

```
Reentrancy in Oracle.createRequest(string,string) (contracts/Oracle.sol#40-84):
        External calls:
        - require(bool,string)(token.transferFrom(msg.sender,owner,requestFee),DEOR transfer Failed.) (contracts/Oracle.sol#47)
        - len = oracles.getOracleCount() (contracts/Oracle.sol#50)
        - selOracle = oracles.getOracleByIndex(orderingOracles[i]) (contracts/Oracle.sol#65)
        - token.transferFrom(selOracle,owner,penaltyForRequest) && now < oracles.getOracleLastActiveTime(selOracle) + 86400 (contracts/Oracle.sol#67)
        State variables written after the call(s):
        - r.quorum[selOracle] = 1 (contracts/Oracle.sol#68)
Reentrancy in Oracle.updateRequest(uint256,string) (contracts/Oracle.sol#87-154):
        External calls:
        - oracles.updateOracleLastActiveTime(msg.sender) (contracts/Oracle.sol#98)
        - oracles.increaseOracleCompleted(msg.sender,responseTime) (contracts/Oracle.sol#104)
        State variables written after the call(s):
        - currRequest.quorum[msg.sender] = 2 (contracts/Oracle.sol#107)
        - currRequest.anwers[msg.sender] = _valueRetrieved (contracts/Oracle.sol#110)
Reentrancy in Oracle.updateRequest(uint256,string) (contracts/Oracle.sol#87-154):
        External calls:
        - oracles.updateOracleLastActiveTime(msg.sender) (contracts/Oracle.sol#98)
        - oracles.increaseOracleCompleted(msg.sender,responseTime) (contracts/Oracle.sol#104)
        - len = oracles.getOracleCount() (contracts/Oracle.sol#114)
        State variables written after the call(s):
        - currRequest.agreedValue = _valueRetrieved (contracts/Oracle.sol#144)
```

[Manual Check] As the external calls are related to token transfer which is token contract internal function that should not posses any significant risk.

[Recommendation] It is recommended to add nonReentrant modifier (https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard-nonReentrant--)

Low/Informational issues:

Missing event:

```
Oracle.setRequestFee(uint256) (contracts/Oracle.sol#30-32) should emit an event for:
        - requestFee = fee (contracts/Oracle.sol#31)
```

[Recommendation] Emit an event for critical parameter changes.

Manual Checks

On-chain

On chain solution consist of Oracle contract which is used to:

- Add/remove oracles from the ecosystem
    - Anyone can create a new oracle, contract owner can remove oracles (used to fight with DDOS attacks on the system)
- Create specific oracle requests
- Aggregates results from multiple oracles utilising random quorum and voting algorithm
- Emits final result from the oracles based on quorum and aggregated responses.

Below there are tests results for 10 oracles and requests creation in the context of gas fees.

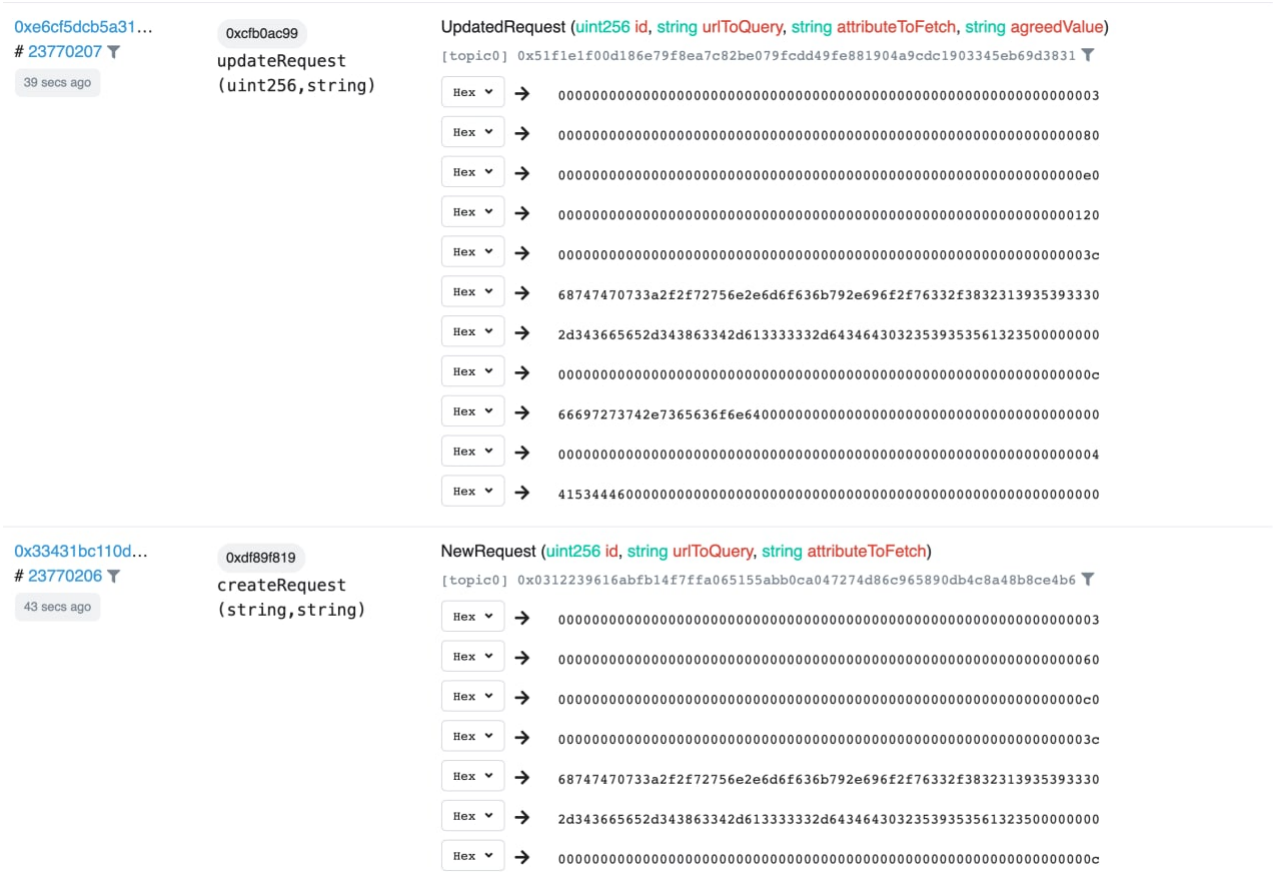
Solc version: 0.6.6	Optimizer enabled: true		Runs: 200	Block limit:
Methods				
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**
DEOR	approve	–	–	43977	10
DEOR	mint	53780	68792	55291	10
Oracle	createRequest	–	–	782599	1
Oracle	newOracle	138783	153783	140283	10
Deployments				**% of limit**
DEOR		–	–	818874	8.6 %
Oracle		–	–	1659830	17.5 %
Oracles		–	–	876819	9.2 %

Gas Fees for oracle creation and requests

Off-chain

Off-chain oracle is a JS/Node application allowing to interact with Oracle smart contract and external API used to gather the data.

We have tested the end to end functionality of the off-chain oracle interacting with on-chain contracts.



Dynamic Tests

We have run fuzzing / property-based testing of Solidity smarts contracts. It was using sophisticated grammar-based fuzzing campaigns based on a contract ABI to falsify user-defined predicates or Solidity assertions.

There were also dynamic tests run on EVM byte code to detect common vulnerabilities including integer underflows, owner-overwrite-to-Ether-withdrawal, and others.

```
The analysis was completed successfully. No issues were detected.
```

No Issues were found.

Deployment & Contract Ownership

The following contract is currently deployed on Ethereum Mainet:

- **DEOR Token - https://etherscan.io/address/0x63726dae7c57d25e90ec829ce9a5c745ffd984d3**

[Recommendation] As Oracle DEOR ecosystem is using Upgradable Oracle contract, it is advisable to deploy Upgradable contract with the ownership transferred to Governance or multi-sig wallet to assure controllable and transparent upgrade process.

Disclaimer

The information appearing in this report is for general purposes only and is not intended to provide any legal security guarantees to any individual or entity. As one review is not enough to provide 100% security against any attacks or bugs, **it is advisable to conduct more reviews or/and audits**.

The report does not provide personalised investment advice or recommendations, especially does not provide advice to conclude any transactions and it does not provide investment, financial, legal or tax advice.

We are not responsible or liable for any loss which results from the report.

The report should not be considered as an investment advice.