CERTIK

Audit Report

Produced by CertiK

for ocean

12th Aug, 2020

CertiK Audit Report For Ocean Protocol

Request Date: 2019-04-03
Revision Date: 2020-08-12
Platform Name: Ethereum

Contents

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Ocean Protocol(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: https://certik.org/

Executive Summary

This report has been prepared for Ocean Protocol to discover issues and vulnerabilities in the source code of their OceanToken smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Vulnerability Classification

CertiK categorizes issues into three buckets based on overall risk levels:

Critical

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Medium

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Low

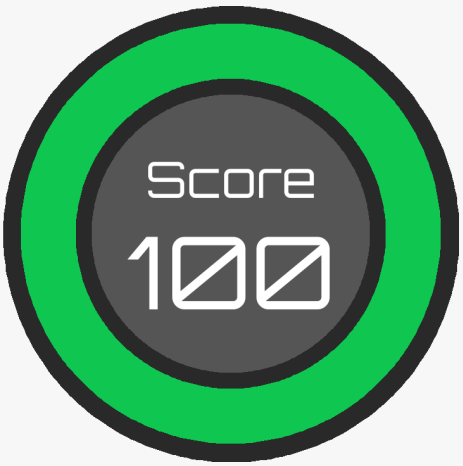
Code implementation does not follow best practices, or uses suboptimal design patterns, which could lead to security vulnerabilities further down the line.

Testing Summary

PASS

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

Aug 12, 2020



Type of Issues

CertiK's smart label engine applied 100% formal verification coverage on the source code. Our team of engineers has scanned the source code using proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues	SWC ID
Integer Overflow/ Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	0	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Using block attributes to generate random numbers is unreliable, as they can be influenced by miners to some degree.	0	SWC-120
"tx.origin" for Authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115

Title	Description	Issues	SWC ID
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default, meaning a malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	SWC-131

Vulnerability Details

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Summary

The Ocean Token is implemented with good engineering quality, and strictly follows the standard ERC20 interface, with minimal set of additional features for central governance and life cycle managements. CertiK does not find any potential security risks with those add-on functionalities, however token holders should still be aware of the administrative authority of the contract owner, who is able to perform critical actions such as pause, mint and kill. On the other hand, given the fact that the token is governed by the Ocean Protocol Foundation (OPF) via a Multisignature wallet, we believe the chance of token getting maliciously manipulated or tampered is low and ignorable. The multisig wallet smart contract is not within the service scope of this audit, thus we cannot provide any assessment or recommendations.

The additional features on top of the standard token mostly focus on data storage and access invokable by token owner. Basically, an array of wallet addresses who hold non-zero balance of Ocean Token is stored in smart contract and will be accessed by owner later for the purpose of migrating the balance from the erc20 token to its mainnet token. The token itself is non-payable (we assume the payable logic is handled on the multisig wallet side) and there is fallback function implemented to revert any value sent, which greatly mitigated the potential risk of being hacked. The contract leans on appropriate standards with minimal storage to fulfill the business requirements and proper intervention mechanism to prevent human errors. We conclude that Ocean Token shall launch in a well-tested and secure state, is not vulnerable to any known antipatterns or bugs, and the risk is likely very low.

Manual Review Notes

Source Code SHA-256 Checksum

- **OceanToken.sol**
  `cf47020508c1de2d37cf792e82227efd2c047c6b349cc75ce61fca4fde2addd8`

Located in https://github.com/oceanprotocol/token/blob/master/contracts/OceanToken.sol at commit hash `b59dfed407cf072a8d91e80c4d3b4beb5320c306`.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File OceanToken.sol

```
1 pragma solidity 0.5.3;
```

Version to compile has the following bug: 0.5.3: TupleAssignmentMultiStackSlotComponents, MemoryArrayCreationOverflow, privateCanBeOverridden, SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor, IncorrectEventSignatureInLibraries, ABIEncoderV2PackedStorage

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

```
30  /*@CTK FAIL "transferFrom to same address"
31      @tag assume_completion
32      @pre from == to
33      @post __post.allowed[from][msg.sender] ==
34  */
```

Raw code location	Line 35-41 in File howtoread.sol

Raw code

```
35  function transferFrom(address from, address to
        ) {
36      balances[from] = balances[from].sub(tokens
37      allowed[from][msg.sender] = allowed[from][
38      balances[to] = balances[to].add(tokens);
39      emit Transfer(from, to, tokens);
40      return true;
41  }
```

Counterexample	This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53             balance: 0x0
54         }
55     }
56
```

Post environment

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

__transfer

12, Aug 2020
1385.76 ms

Line 50-56 in File OceanToken.sol

```
50    /*@CTK _transfer
51      @tag assume_completion
52      @pre msg.sender != _to
53      @post _to != address(0)
54      @post __post._balances[msg.sender] == _balances[msg.sender] - _value
55      @post __post._balances[_to] == _balances[_to] + _value
56     */
```

Line 57-69 in File OceanToken.sol

```
57    function transfer(
58        address _to,
59        uint256 _value
60    )
61    public
62    returns (bool)
63    {
64        bool success = super.transfer(_to, _value);
65        if (success) {
66            updateTokenHolders(msg.sender, _to);
67        }
68        return success;
69    }
```

The code meets the specification.

Formal Verification Request 2

transfer_from

12, Aug 2020
1872.25 ms

Line 78-87 in File OceanToken.sol

```
78    /*@CTK transfer_from
79      @tag assume_completion
80      @pre _from != _to
81      @post _to != address(0)
82      @post _value <= _allowed[_from][msg.sender]
83      @post __post._balances[_from] == _balances[_from] - _value
84      @post __post._balances[_to] == _balances[_to] + _value
85      @post __post._allowed[_from][msg.sender] ==
86      _allowed[_from][msg.sender] - _value
87     */
```

Line 88-101 in File OceanToken.sol

```
function transferFrom(
    address _from,
    address _to,
    uint256 _value
)
public
returns (bool)
{
    bool success = super.transferFrom(_from, _to, _value);
    if (success) {
        updateTokenHolders(_from, _to);
    }
    return success;
}
```

✅ The code meets the specification.

Formal Verification Request 3

getAccountsLength

📅 12, Aug 2020
⏱ 41.39 ms

Line 145-149 in File OceanToken.sol

```
145 /*@CTK getAccountsLength
146   @tag assume_completion
147   @post _owner == msg.sender
148   @post __return == accounts.length
149  */
```

Line 150-157 in File OceanToken.sol

```
150 function getAccountsLength()
151 external
152 view
153 onlyOwner
154 returns (uint256)
155 {
156     return accounts.length;
157 }
```

✅ The code meets the specification.

Formal Verification Request 4

tryToAddTokenHolder

📅 12, Aug 2020
⏱ 7.08 ms

Line 183-188 in File OceanToken.sol

```
183 /*@CTK tryToAddTokenHolder
184   @tag assume_completion
```

```
185     @pre !tokenHolders[account] && _balances[account] > 0
186     @post __post.accounts[accounts.length] == account
187     @post __post.tokenHolders[account]
188    */
```

Line 189-199 in File OceanToken.sol

```
189    function tryToAddTokenHolder(
190        address account
191    )
192    private
193    {
194        if (!tokenHolders[account] && super.balanceOf(account) > 0)
195        {
196            accounts.push(account);
197            tokenHolders[account] = true;
198        }
199    }
```

✅ The code meets the specification.

Formal Verification Request 5

updateTokenHolders

📅 12, Aug 2020
⏱ 75.91 ms

Line 206-215 in File OceanToken.sol

```
206    /*@CTK updateTokenHolders
207      @tag assume_completion
208      @pre sender != receiver
209      @pre !tokenHolders[sender] && _balances[sender] > 0
210      @pre !tokenHolders[receiver] && _balances[receiver] > 0
211      @post __post.accounts[accounts.length] == sender
212      @post __post.tokenHolders[sender]
213      @post __post.accounts[accounts.length + 1] == receiver
214      @post __post.tokenHolders[receiver]
215     */
```

Line 216-224 in File OceanToken.sol

```
216    function updateTokenHolders(
217        address sender,
218        address receiver
219    )
220    private
221    {
222        tryToAddTokenHolder(sender);
223        tryToAddTokenHolder(receiver);
224    }
```

✅ The code meets the specification.

Formal Verification Request 6

Migrations

12, Aug 2020
11.23 ms

Line 7-9 in File Migrations.sol

```
7   /*@CTK Migrations
8     @post __post.owner == msg.sender
9    */
```

Line 10-12 in File Migrations.sol

```
10  constructor() public {
11      owner = msg.sender;
12  }
```

The code meets the specification.

Formal Verification Request 7

setCompleted

12, Aug 2020
12.72 ms

Line 18-21 in File Migrations.sol

```
18  /*@CTK setCompleted
19    @pre msg.sender == owner
20    @post __post.last_completed_migration == completed
21   */
```

Line 22-24 in File Migrations.sol

```
22  function setCompleted(uint completed) public restricted {
23      last_completed_migration = completed;
24  }
```

The code meets the specification.

Source Code with CertiK Labels

File OceanToken.sol

```
pragma solidity 0.5.3;

import "openzeppelin-solidity/contracts/token/ERC20/ERC20Capped.sol";
import "openzeppelin-solidity/contracts/token/ERC20/ERC20Detailed.sol";
import "openzeppelin-solidity/contracts/token/ERC20/ERC20Pausable.sol";
import "openzeppelin-solidity/contracts/ownership/Ownable.sol";

/**
 * @title Ocean Protocol ERC20 Token Contract
 * @author Ocean Protocol Team
 * @dev Implementation of the Ocean Token.
 */
contract OceanToken is Ownable, ERC20Pausable, ERC20Detailed, ERC20Capped {

    using SafeMath for uint256;

    uint8 constant DECIMALS = 18;
    uint256 constant CAP = 1410000000;
    uint256 TOTALSUPPLY = CAP.mul(uint256(10) ** DECIMALS);

    // keep track token holders
    address[] private accounts = new address[](0);
    mapping(address => bool) private tokenHolders;

    /**
     * @dev Ocean Token constructor
     * @param contractOwner refers to the owner of the contract
     */
    constructor(
        address contractOwner
    )
    public
    ERC20Detailed('Ocean Token', 'OCEAN', DECIMALS)
    ERC20Capped(TOTALSUPPLY)
    Ownable()
    {
        addPauser(contractOwner);
        renouncePauser();
        addMinter(contractOwner);
        renounceMinter();
        transferOwnership(contractOwner);
    }

    /**
     * @dev transfer tokens when not paused (pausable transfer function)
     * @param _to receiver address
     * @param _value amount of tokens
     * @return true if receiver is illegible to receive tokens
     */
    /*@CTK _transfer
      @tag assume_completion
      @pre msg.sender != _to
      @post _to != address(0)
      @post __post._balances[msg.sender] == _balances[msg.sender] - _value
```

```
     @post __post._balances[_to] == _balances[_to] + _value
    */
    function transfer(
        address _to,
        uint256 _value
    )
    public
    returns (bool)
    {
        bool success = super.transfer(_to, _value);
        if (success) {
            updateTokenHolders(msg.sender, _to);
        }
        return success;
    }

    /**
     * @dev transferFrom transfers tokens only when token is not paused
     * @param _from sender address
     * @param _to receiver address
     * @param _value amount of tokens
     * @return true if receiver is illegible to receive tokens
     */
    /*@CTK transfer_from
      @tag assume_completion
      @pre _from != _to
      @post _to != address(0)
      @post _value <= _allowed[_from][msg.sender]
      @post __post._balances[_from] == _balances[_from] - _value
      @post __post._balances[_to] == _balances[_to] + _value
      @post __post._allowed[_from][msg.sender] ==
      _allowed[_from][msg.sender] - _value
     */
    function transferFrom(
        address _from,
        address _to,
        uint256 _value
    )
    public
    returns (bool)
    {
        bool success = super.transferFrom(_from, _to, _value);
        if (success) {
            updateTokenHolders(_from, _to);
        }
        return success;
    }

    /**
     * @dev retrieve the address & token balance of token holders (each time retrieve
         partial from the list)
     * @param _start index
     * @param _end index
     * @return array of accounts and array of balances
     */
    function getAccounts(
        uint256 _start,
        uint256 _end
```

```
    )
    external
    view
    onlyOwner
    returns (address[] memory, uint256[] memory)
    {
        require(
            _start <= _end && _end < accounts.length,
            'Array index out of bounds'
        );

        uint256 length = _end.sub(_start).add(1);

        address[] memory _tokenHolders = new address[](length);
        uint256[] memory _tokenBalances = new uint256[](length);

        for (uint256 i = _start; i <= _end; i++)
        {
            address account = accounts[i];
            uint256 accountBalance = super.balanceOf(account);
            if (accountBalance > 0)
            {
                _tokenBalances[i] = accountBalance;
                _tokenHolders[i] = account;
            }
        }

        return (_tokenHolders, _tokenBalances);
    }

    /**
     * @dev get length of account list
     */
    /*@CTK getAccountsLength
      @tag assume_completion
      @post _owner == msg.sender
      @post __return == accounts.length
     */
    function getAccountsLength()
    external
    view
    onlyOwner
    returns (uint256)
    {
        return accounts.length;
    }

    /**
     * @dev kill the contract and destroy all tokens
     */
    function kill()
    external
    onlyOwner
    {
        selfdestruct(address(uint160(owner())));
    }

    /**
```

```
     * @dev fallback function prevents ether transfer to this contract
     */
    function()
    external
    payable
    {
        revert('Invalid ether transfer');
    }

    /*
     * @dev tryToAddTokenHolder try to add the account to the token holders structure
     * @param account address
     */
    /*@CTK tryToAddTokenHolder
      @tag assume_completion
      @pre !tokenHolders[account] && _balances[account] > 0
      @post __post.accounts[accounts.length] == account
      @post __post.tokenHolders[account]
     */
    function tryToAddTokenHolder(
        address account
    )
    private
    {
        if (!tokenHolders[account] && super.balanceOf(account) > 0)
        {
            accounts.push(account);
            tokenHolders[account] = true;
        }
    }

    /*
     * @dev updateTokenHolders maintains the accounts array and set the address as a
       promising token holder
     * @param sender address
     * @param receiver address.
     */
    /*@CTK updateTokenHolders
      @tag assume_completion
      @pre sender != receiver
      @pre !tokenHolders[sender] && _balances[sender] > 0
      @pre !tokenHolders[receiver] && _balances[receiver] > 0
      @post __post.accounts[accounts.length] == sender
      @post __post.tokenHolders[sender]
      @post __post.accounts[accounts.length + 1] == receiver
      @post __post.tokenHolders[receiver]
     */
    function updateTokenHolders(
        address sender,
        address receiver
    )
    private
    {
        tryToAddTokenHolder(sender);
        tryToAddTokenHolder(receiver);
    }
}
```

File Migrations.sol

```
pragma solidity >=0.4.21 <0.6.0;

contract Migrations {
    address public owner;
    uint public last_completed_migration;

    /*@CTK Migrations
      @post __post.owner == msg.sender
     */
    constructor() public {
        owner = msg.sender;
    }

    modifier restricted() {
        if (msg.sender == owner) _;
    }

    /*@CTK setCompleted
      @pre msg.sender == owner
      @post __post.last_completed_migration == completed
     */
    function setCompleted(uint completed) public restricted {
        last_completed_migration = completed;
    }

    function upgrade(address new_address) public restricted {
        Migrations upgraded = Migrations(new_address);
        upgraded.setCompleted(last_completed_migration);
    }
}
```