

SMART CONTRACT AUDIT REPORT

For

**Ollie NFT artwork**

**Prepared By**: SFI Team **Prepared for**: ollie team

**Prepared on**: 11/12/2021

**Table of Content**

* Disclaimer
* Overview of the audit
* Attacks made to the contract
* Good things in smart contract
* Critical vulnerabilities found in the contract
* High vulnerabilities found in the contract
* Medium vulnerabilities found in the contract
* Low severity vulnerabilities found in the contract
* Very Low severity vulnerabilities found in the contract
* Notes
* Testing proves
* Unified Modeling Language (UML)
* Functions signature
* Automatic general report
* Summary of the audit
* **Disclaimer**

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against the team on the basis of what it says or doesn’t say, or how team produced it, and it is important for you to conduct your own independent investigations before making any decisions. team go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Saferico and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SaferICO ) owe no duty of care towards you or any other person, nor does Saferico make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Saferico hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Saferico hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saferico, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

* **Overview of the audit**

The project has 3 file. It contains approx 336 lines of Solidity code. Most of the functions and state variables are well commented on using the Nat spec documentation, but that does not create any vulnerability.

* **Attacks made to the contract**

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices automatically.

1. Unit tests passing.

2. Compilator warnings;

3. Race Conditions. Reentrancy. Cross-function Race Conditions. Pitfalls in Race Condition solutions;

4. Possible delays in data delivery;

5. Transaction-Ordering Dependence (front running);

6. Timestamp Dependence;

7. Integer Overflow and Underflow;

8. DoS with (unexpected) Revert;

9. DoS with Block Gas Limit

10. Call Depth Attack. Not relevant in modern ethereum network

11. Methods execution permissions;

12. Oracles calls;

13. Economy model. It’s important to forecast scenarios when a user is provided with additional economic motivation or faced with limitations. If application logic is based on incorrect economy model, the application will not function correctly and participants will incur financial losses. This type of issue is most often found in bonus rewards systems.  
14. The impact of the exchange rate on the logic;  
15. Private user data leaks.

# Good things in smart contract

# Compiler version is static: -

=> In this file, you have put “pragma solidity 0.8.0;” which is a good way to define the compiler version.

pragma solidity 0.8.0;

### openzeppelin library: -

Ollie is using openzeppelin library it is a good thing. All contract is based on openzeppelin library which develops by professional developers and it is one of the most secured library in the blockchain industry

import "@openzeppelin/contracts/access/Ownable.sol";

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

### Ownable library : -

* + Here you ollie token using ownable library, Initializes the contract setting the deployer as the initial owner

abstract contract Ownable is Context {  
 address private \_owner;  
  
 event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);  
  
 /\*\*  
 \* @dev Initializes the contract setting the deployer as the initial owner.  
 \*/  
 constructor() {  
 \_setOwner(\_msgSender());  
 }  
  
 function owner() public view virtual returns (address) {  
 return \_owner;  
 }

* + Here you ollie token using ERC721 library this Implementation of Non-Fungible Token Standard, including the Metadata extension

contract ERC721 is Context, ERC165, IERC721, IERC721Metadata {  
 using Address for address;  
 using Strings for uint256;  
  
 // Token name  
 string private \_name;

// Token symbol  
 string private \_symbol;  
  
 // Mapping from token ID to owner address  
 mapping(uint256 => address) private \_owners;  
  
 // Mapping owner address to token count  
 mapping(address => uint256) private \_balances;  
  
 // Mapping from token ID to approved address  
 mapping(uint256 => address) private \_tokenApprovals;  
  
 // Mapping from owner to operator approvals  
 mapping(address => mapping(address => bool)) private \_operatorApprovals;

* + **Critical vulnerabilities found in the contract**

**There not Critical severity vulnerabilities found**

* + **High vulnerabilities found in the contract**

# There not High severity vulnerabilities found

* + **Medium vulnerabilities found in the contract**

# There not Medium severity vulnerabilities found

* + **Low vulnerabilities found in the contract**

# There not Low severity vulnerabilities found

* + **V. Low vulnerabilities found in the contract**

# #Similar variable names:

\_name = name\_;

\_symbol = symbol\_;

In detail

ERC721.(string,string) : Variables have very similar names "\_symbol" and "symbol\_". Note: Modifiers are currently not considered by this static analysis.

# block.timestamp:

function nextToken() internal override ensureAvailability returns (uint256) {  
 uint256 maxIndex = totalSupply() - tokenCount();  
 uint256 random = uint256(keccak256(  
 abi.encodePacked(  
 msg.sender,  
 block.coinbase,  
 block.difficulty,  
 block.gaslimit,  
 block.timestamp  
 )

In detail

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.

That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

* + **Notes**

# #Gas cost:

function name() public view virtual override returns (string memory) {

return \_name;

}

In detail

Gas requirement of function ERC721.name is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

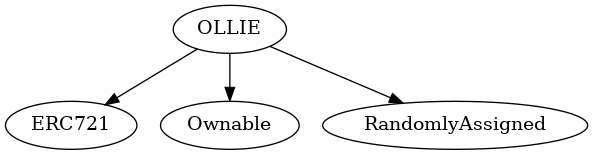
# Testing proves:

# Check for security

1. **SOLIDITY STATIC ANALYSIS**

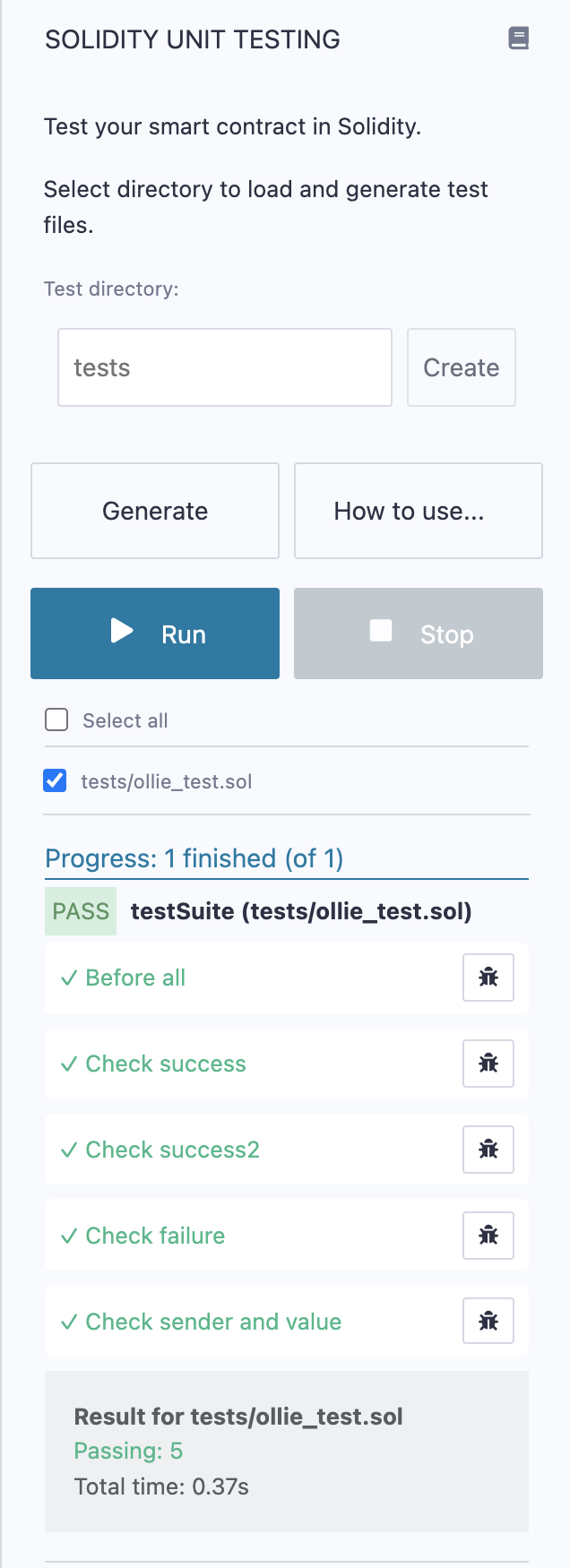
# 

# Inheritance graph

****

1. **SOLIDITY UNIT TESTING CODE & RESULTS**

// SPDX-License-Identifier: GPL-3.0  
   
pragma solidity >=0.4.22 <0.9.0;  
  
// This import is automatically injected by Truffle  
import " Truffle \_tests.sol";   
  
// This import is required to use custom transaction context  
// Although it may fail compilation in 'Solidity Compiler' plugin  
  
import " Truffle\_accounts.sol";  
import "../ollie.sol";  
  
// File name has to end with '\_test.sol', this file can contain more than one testSuite contracts  
contract testSuite {  
  
 /// 'beforeAll' runs before all other tests  
 /// More special functions are: 'beforeEach', 'beforeAll', 'afterEach' & 'afterAll'  
 function beforeAll() public {  
 // <instantiate contract>  
 Assert.equal(uint(1), uint(1), "1 should be equal to 1");  
 }  
  
 function checkSuccess() public {  
 // Use 'Assert' methods:  
 Assert.ok(2 == 2, 'should be true');  
 Assert.greaterThan(uint(2), uint(1), "2 should be greater than to 1");  
 Assert.lesserThan(uint(2), uint(3), "2 should be lesser than to 3");  
 }  
  
 function checkSuccess2() public pure returns (bool) {  
 // Use the return value (true or false) to test the contract  
 return true;  
 }  
   
 function checkFailure() public {  
 Assert.notEqual(uint(1), uint(2), "1 should not be equal to 1");  
 }  
  
   
 /// #sender: account-1  
 /// #value: 100  
 function checkSenderAndValue() public payable {  
 // account index varies 0-9, value is in wei  
 Assert.equal(msg.sender, TestsAccounts.getAccount(1), "Invalid sender");  
 Assert.equal(msg.value, 100, "Invalid value");  
 }  
}



# Call graph

# 

# Unified Modeling Language (UML)

# 

# Function Signature

743976a0 => \_baseURI()  
6ecd2306 => mint(uint8)  
3af32abf => isWhitelisted(address)  
3c952764 => setOnlyWhitelisted(bool)  
edec5f27 => whitelistUsers(address[])  
438b6300 => walletOfOwner(address)  
c87b56dd => tokenURI(uint256)  
44a0d68a => setCost(uint256)  
f8f53e6f => setmaxMintAmount(uint8)  
55f804b3 => setBaseURI(string)  
da3ef23f => setBaseExtension(string)  
02329a29 => pause(bool)  
3ccfd60b => withdraw()  
2f745c59 => tokenOfOwnerByIndex(address,uint256)  
69025b5f => \_addTokenToOwnerEnumeration(address,uint256)  
68df0d53 => \_removeTokenFromOwnerEnumeration(address,uint256)  
4cbb4a0a => \_removeTokenFromAllTokensEnumeration(uint256)  
e03d890b => \_addTokenToAllTokensEnumeration(uint256)  
cad3be83 => \_beforeTokenTransfer(address,address,uint256)

* **Automatic general report**

| File Name | SHA-1 Hash |  
|-------------|--------------|  
| /Users/macbook/Desktop/smart contracts/ollie.sol | aebff6744707c29085771e3ba803c234e133d4d8 |  
  
 Contracts Description Table  
  
| Contract | Type | Bases | | |  
|:----------:|:-------------------:|:----------------:|:----------------:|:---------------:|  
| └ | \*\*Function Name\*\* | \*\*Visibility\*\* | \*\*Mutability\*\* | \*\*Modifiers\*\* |  
||||||  
| \*\*OLLIE\*\* | Implementation | ERC721, Ownable, RandomlyAssigned |||  
| └ | <Constructor> | Public ❗️ | 🛑 | ERC721 RandomlyAssigned |  
| └ | \_baseURI | Internal 🔒 | | |  
| └ | mint | Public ❗️ | 💵 |NO❗️ |  
| └ | isWhitelisted | Public ❗️ | |NO❗️ |  
| └ | setOnlyWhitelisted | Public ❗️ | 🛑 | onlyOwner |  
| └ | whitelistUsers | Public ❗️ | 🛑 | onlyOwner |  
| └ | walletOfOwner | Public ❗️ | |NO❗️ |  
| └ | tokenURI | Public ❗️ | |NO❗️ |  
| └ | setCost | Public ❗️ | 🛑 | onlyOwner |  
| └ | setmaxMintAmount | Public ❗️ | 🛑 | onlyOwner |  
| └ | setBaseURI | Public ❗️ | 🛑 | onlyOwner |  
| └ | setBaseExtension | Public ❗️ | 🛑 | onlyOwner |  
| └ | pause | Public ❗️ | 🛑 | onlyOwner |  
| └ | withdraw | Public ❗️ | 💵 | onlyOwner |  
| └ | tokenOfOwnerByIndex | Public ❗️ | |NO❗️ |  
| └ | \_addTokenToOwnerEnumeration | Private 🔐 | 🛑 | |  
| └ | \_removeTokenFromOwnerEnumeration | Private 🔐 | 🛑 | |  
| └ | \_removeTokenFromAllTokensEnumeration | Private 🔐 | 🛑 | |  
| └ | \_addTokenToAllTokensEnumeration | Private 🔐 | 🛑 | |  
| └ | \_beforeTokenTransfer | Internal 🔒 | 🛑 | |  
  
 Legend  
  
| Symbol | Meaning |  
|:--------:|-----------|  
| 🛑 | Function can modify state |  
| 💵 | Function is payable |

# Summary of the Audit

According to automatically test, the customer`s solidity smart contract is **Secured**.

The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

The test found 0 critical, 0 high, 0 medium, 0 low, 2 Very low issues, and 1 note.