CERTIK

Trister World tCard

Security Assessment

March 15th, 2021

For :

Trister

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	Trister
Description	Trister SmartNFT（tCard）
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	b99a9097a48e42c0b33a444efe817c6345571501

Audit Summary

Delivery Date	March. 15th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Mar. 8th, 2021 - Mar. 12th, 2021 & Mar. 15th, 2021

Vulnerability Summary

Total Issues	10
🔴 **Total Critical**	0
🟡 **Total Major**	0
🔵 **Total Minor**	3
🟢 **Total Informational**	7

Executive Summary

This report has been prepared for **Trister** smart contract to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, 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.

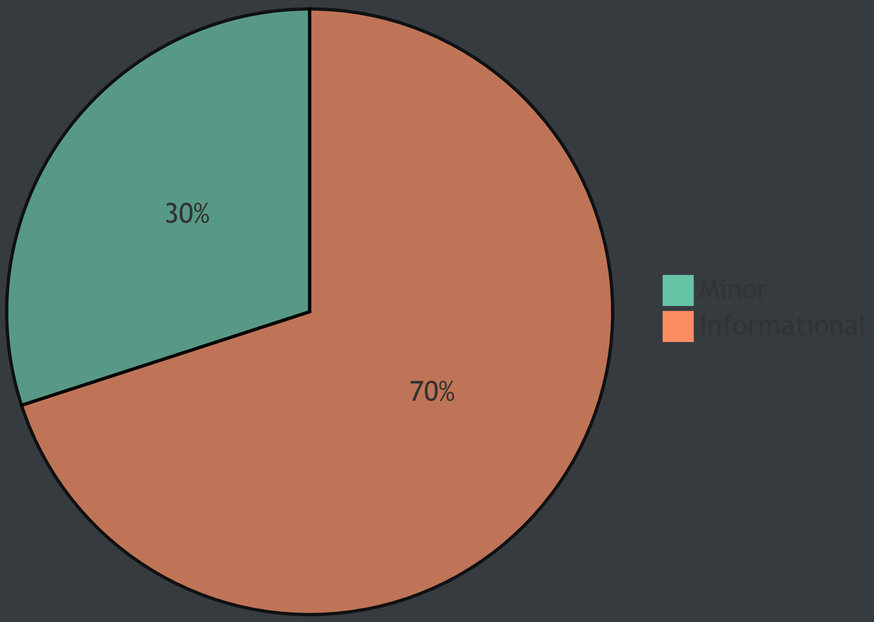
To bridge the trust gap between administrator and users, administrator needs to express a sincere attitude with the consideration of the administrator team’s anonymousness. The administrator has the responsibility to notify users with the following capability of the administrator:

- Owner has privilege to mint tokens.

File in Scope

Contract	SHA-256 Checksum	ID
CST	**tlc-sc-tcard-release/contracts/Strings.sol**	4703e8d8a4a2240e214720a757d2cb3f19472c2d6e21dc1563ba6bf95a36ce9f
CIF	**tlc-sc-tcard-release/contracts/IFactoryERC721.sol**	7ddd3015fae444a575df1101ca15b7526ecaeb7ea3d51d81d18d316723837613
CTD	**tlc-sc-tcard-release/contracts/ERC721Tradable.sol**	5929a395b3def36f9efec85733011b99f0146a86063e9bc8efd05474d4794190
SIF	**tlc-sc-seller-release/contracts/IFactoryERC721.sol**	9fab862db1364960b9db6b5bbacf12c46fd90c3aaa2435e4ff2aab054afe3c6d
SMD	**tlc-sc-seller-release/contracts/TCardSellerMetadata.sol**	e9ec9bd3bd1e466c3bf8cfa4ac8bf3b14e0ad294d5ff6f88b5f71e52abc741a1
SCF	**tlc-sc-seller-release/contracts/TCardSellerConfig.sol**	450d1bc1b793df5af25b3d568d6e46bdc514b24dc170c122ddaa6293ff510878
SST	**tlc-sc-seller-release/contracts/TCardSellerStock.sol**	403d269d1ade3d498fbca2920f497eeaf6a2177c45746a55d0ad091365f7ea8f
SLR	**tlc-sc-seller-release/contracts/TCardSeller.sol**	d9f7cd9d2261b374431a378947669ff4c379641efa6d799d2a810e9eb7f187a7

Findings



ID	Title	Type	Severity	Resolved
SIF-01	Unlocked Compiler Version Declaration & Different Versions of Solidity Used	Language Specific Issue	🟢 Informational	❗
SCF-01	Missing Emit Events	Optimization	🔵 Minor	❗
SCF-02	Missing Zero Address Checks	Logical Issue	🔵 Minor	❗
CIF-01	Idle Interface	Dead Code	🟢 Informational	❗
CTD-01	Owner's Minting Privilege	Logical Issue	🔵 Minor	❗
CTD-02	Boolean Equality	Optimization	🟢 Informational	❗
SST-01	Idle Variable	Optimization	🟢 Informational	❗
SST-02	State Variables That Could Be Declared `constant`	Gas Optimization	🟢 Informational	❗
SSL-01	Events Should Add `indexed` Keyword	Gas Optimization	🟢 Informational	❗
SSL-02	Checks-Effects-Interactions Pattern Violated	Logical Issue	🔵 Minor	❗

SIF-01: Unlocked Compiler Version Declaration & Different Versions of Solidity Used

Type	Severity	Location
Language Specific Issue	🟢 Infomational	tlc-sc-seller-release/contracts/IFactoryERC721.sol L1

Description:

The compiler version utilized throughout the project uses the “^” prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts. `Solidity ^0.5.0` is declared in `tlc-sc-seller-release/contracts/IFactoryERC721.sol` while all other files from the same folder use `Solidity ^0.5.11`. The compiler version should be consistent throughout the codebase.

Recommendation:

We recommend locking the compiler the lowest possible version that supports all the capabilities wished by the codebase.

Alleviation:

No alleviation.

SCF-01: Missing Emit Events

Type	Severity	Location
Optimization	● Minor	TCardSellerConfig.sol L44,L56,L60,L68; TCardSeller.sol L120

Description:

Some functions alter key parameters of the contract. Function that affect the status of sensitive variables should be able to emit events as notifications to customers.

For examples:
In `TCardSellerConfig.sol` :
`setTimeInterval()`
`setNFTAddress()`
`setTokenAddress()`
`setPayeeAddress()`

In `TCardSeller.sol` :
`setEnabled()`

Recommendation:

We recommend declaring events then emitting them in respective setter functions.
For example, revise `setEnabled()` at `TCardSellerConfig.sol, L56` as follows:

```
event NFTAddress(address indexed _nftAddress);

function setNFTAddress(address _nftAddress) public onlyOwner isModifiable {
    nftAddress = _nftAddress;
    emit NFTAddress(_nftAddress);
}
```

Alleviation:

No alleviation.

SCF-02: Missing Zero Address Checks

Type	Severity	Location
Logical Issue	● Minor	TCardSellerConfig.sol L56,L60,L68;TCardSeller.sol 203

Description:

Some functions use addresss without validation. Discretion is advised when dealing with them.

For examples:
In `TCardSellerConfig.sol` :
`setNFTAddress()`
`setTokenAddress()`
`setPayeeAddress()`

In `TCardSeller.sol` :
`withdraw()`

Recommendation:

We recommend using `require` statements to make sure the address arguments are not zero addresses.
For example in `setNFTAddress()` at TCardSellerConfig.sol, L60

```
        function setTokenAddress(address _tokenAddress)
        public
        onlyOwner
        isModifiable
        {
            require(_tokenAddress != address(0), "Invalid Address Input!");
            tokenAddress = _tokenAddress;
        }
```

Alleviation:

No alleviation.

CIF-01: Idle Interface

Type	Severity	Location
Dead Code	🟢 Informational	tlc-sc-tcard-release/contracts/IFactoryERC721.sol

Description:

`IFactoryERC721` is never used anywhere in tlc-sc-tcard contracts.

Recommendation:

We recommend removing the whole file unless it is needed in further development.

Alleviation:

No alleviation.

CTD-01: Owner's Minting Privilege

Type	Severity	Location
Logical Issue	● Minor	ERC721Tradable.sol, L84,L98

Description:

The administrator holds minting capability as stated in `mintTo()` at `ERC721Tradable.sol` .

Recommendation:

To bridge the trust gap between administrator and users, administrator needs to express a sincere attitude with the consideration of the administrator team's anonymity.
We would like to inquire about further development plans, concerning potential restrictions on minters.

Alleviation:

(**Trister-Response**)-This is because the cost of NFT casting is too high, and the customer cannot determine the total circulation of NFT for the time being, so we often grant the owner direct casting in the code.

CTD-02: Boolean Equality

Type	Severity	Location
Optimization	● Informational	ERC721Tradable.sol, L85,L99

Description:

Boolean constants can be used directly in conditionals and do not need to be compared to true or false.

```
require(canMint == true, "ERC721: canMint is false");
```

Recommendation:

We recommend using the boolean `canMint` directly as follows:

```
require(canMint,"ERC721: canMint is false");
```

Alleviation:

No alleviation.

SST-01: Idle Variable

Type	Severity	Location
Dead Code	● Informational	TCardSellerStock.sol, L10

Description:

The variable `maxId` is never used anywhere.

Recommendation:

We recommend removing it for gas optimization.

Alleviation:

No alleviation.

SST-02: State Variables That Could Be Declared `constant`

Type	Severity	Location
Gas Optimization	● Informational	TCardSellerStock.sol, L11

Description:

The variable `MAX_UINT256` is never altered beyond declaration.

Recommendation:

We recommend declaring `MAX_UINT256` as `constant` for gas optimization.

Alleviation:

No alleviation.

SSL-01: Events Should Add `indexed` Keyword

Type	Severity	Location
Gas Optimization	🟢 Informational	ERC721Tradable.sol, L34,L43,L54,L65; TCardSeller.sol, L34,L45

Description:

It is often preferable to add `indexed` for log-keeping and gas optimization sake in `event` , especially when arrays are involved.

For examples, in `ERC721Tradable.sol` :
`address _to` in `Mint()`
`address _to, uint256 _newTokenId` in `MintByTokenId()`

Recommendation:

We recommend declaring some of these event parameters as `indexed` .
For example, `event Mint()` at ERC721Tradable.sol, L34 could be revised as follows

```
event Mint(
    address indexed _to
);
```

Alleviation:

No alleviation.

SSL-02: Checks-Effects-Interactions Pattern Violated

Type	Severity	Location
Logical Issue	● Minor	TCardSeller.sol, L79

Description:

NFT is minted before validation of sale, which violates checks-effects-interactions pattern.
Reference: https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html

Recommendation:

It is recommended to follow checks-effects-interactions pattern for cases like this. It shields public/external functions from re-entrancy attacks. It’s always a good practice to follow this pattern.

For example in `buyWithToken()` at `TCardSeller.sol` , L79, put the `require` check before `nft.mintByTokenId()` :

```
function buyWithToken()
public
isEnabled
isSellingTime
checkStock
checkBalance
{
    ...

    // transfer token to payeeAddress;
    (bool success, bytes memory data) =
        tokenAddress.call(
            abi.encodeWithSelector(
                SELECTOR,
                msg.sender,
                payeeAddress,
                nftPrice
            )
        );
    require(
        success && (data.length == 0 || abi.decode(data, (bool))),
        "transferFrom: TRANSFER_FAILED"
    );

    nft.mintByTokenId(msg.sender, tokenId);

    ...
}
```

Alleviation:

No alleviation.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

✓ : Issue resolved

ⓘ : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

ⓘ✓ : Issue partially resolved. Not all instances of an issue was resolved.