HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TravelCare
Date: March 1st, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for TravelCare.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token
Platform	BSC
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Deployed contract	https://bscscan.com/address/0x826e5ec70dbc5607ff9218011fbb97f9a8d97953#code
Technical Documentation	YES
JS tests	NO
Website	https://travelcare.io/travel-token/
Timeline	23 February 2022 - 1 March 2022
Changelog	1 March 2022 - Initial Audit

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by TravelCare (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between February 23rd, 2022 - March 1st, 2022.

Scope

The scope of the project is smart contracts on BSC chain with address:
Contract:
https://bscscan.com/address/0x826e5ec70dbc5607ff9218011fbb97f9a8d97953#code
Technical Documentation: Yes
JS tests: No
Contracts:
TravelCare.sol
IERC20Metadata.sol
Context.sol
ERC20.sol
IERC20.sol
Ownable.sol

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

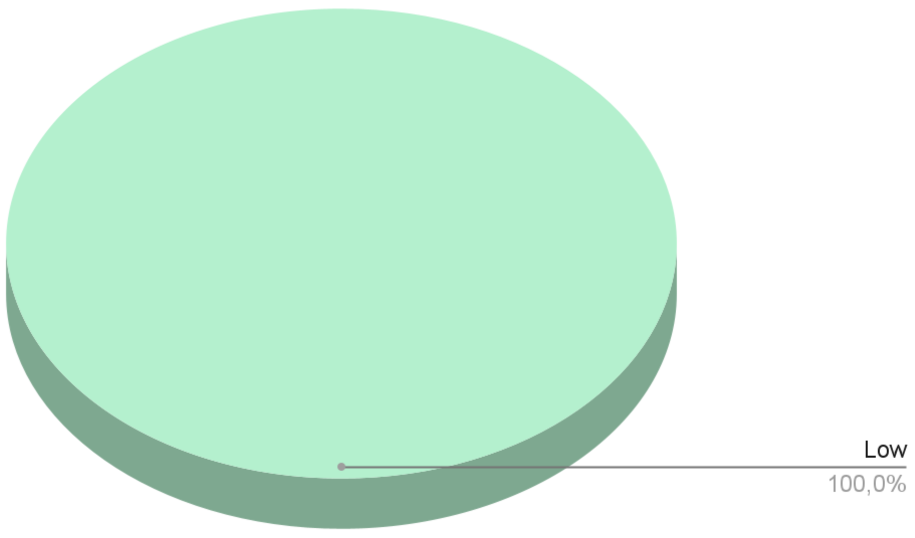
According to the assessment, the Customer's smart contracts are well-secured.

Insecure
Poor secured
Secured
Well-secured
You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril, SmartCheck, Solgraph, Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **9** low severity issues.

Graph 1. The distribution of vulnerabilities after the audit.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Audit overview

🟥🟥🟥🟥 Critical

No critical issues were found.

🟨🟨🟨 High

No high severity issues were found.

🟩🟩 Medium

No high severity issues were found.

🟦 Low

1. External imports used but files are present locally.
Contract *TravelCare.sol*

```
import { ERC20 } from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import { IERC20 } from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

import { Ownable } from "@openzeppelin/contracts/access/Ownable.sol";
```

Recommendation: Change to local imports:

```
import { ERC20 } from "./ERC20.sol";

import { IERC20 } from "./IERC20.sol";

import { Ownable } from "./Ownable.sol";
```

2. Contract *ERC20.sol*, function *transferFrom*
It’s better to execute check and update allowance before *_transfer()* function call.
Recommendation: Move *_transfer()* to the end of function (after *require* and *unchecked*).

3. Ambiguous tokens amount for **Pink Flamingo** holders.
There is required amount of **210** tokens in the documentation but **1000** tokens amount is used in code.
Recommendation: Review your requirements and update either code or documentation.

4. Empty function invocation

_beforeTokenTransfer function is empty therefore when its executed - nothing happens.
This functions is invoked in: *_transfer(), _mint(), _burn().*

Recommendation:

To save gas on contract deployment and execution we recommend to remove this function and all its invocations.

5. Call of *changeRewardRequirements()* method affects only the **future** reward type assignments and does not change the reward types for **current** users rewards.

Recommendation:
Pay attention to this logic if it is supposed to work in such way. Fix if required. Maybe you want to update user reward type each time you update the requirements for any of reward type.

6. Not latest solidity version used.

Recommendation: The latest version is 0.8.12. It's recommended to use it.

7. Unused function
Contract *Context.sol*. Function ***_msgData*** is unused.
Recommendation: Delete ***_msgData*** function from the contract.

8. Code duplication
The code from *TravelCare.sol _transfer* function

```
    require(

        sender != address(0),

        "ERC20::_transfer: transfer from the zero address"

    );

    require(

        recipient != address(0),

        "ERC20::_transfer: transfer to the zero address"

    );
```

duplicates the code from *ERC20.sol _transfer* function

```
require(sender != address(0), "ERC20: transfer from the zero address");

require(recipient != address(0), "ERC20: transfer to the zero address");
```

Recommendation: Remove code duplication from *TravelCare.sol _transfer* function.

9. Some functions are declared as ***public*** instead of being declared ***external***.

- *ERC20.name()*
- *ERC20.symbol()*
- *ERC20.decimals()*
- *ERC20.totalSupply()*
- *ERC20.balanceOf()*
- *ERC20.transfer()*
- *ERC20.allowance()*
- *ERC20.approve()*
- *ERC20.transferFrom()*
- *ERC20.increaseAllowance(*
- *ERC20.decreaseAllowance()*
- *Ownable.renounceOwnership()*
- *Ownable.transferOwnership()*

Recommendation: public functions that are never called by the contract should be declared ***external*** to save gas.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **9** low severity issues.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.