Smart Contract Security Audit Report

For

Acara

Date Issued: July 9, 2023

Version: v1.0

Confidentiality Level: Public

Contents

1 Abstract

This report was prepared for Acara smart contract to identify issues and vulnerabilities in its smart contract source code. A thorough examination of Acara smart contracts was conducted through timely communication with Acara, static analysis using multiple audit tools and manual auditing of their smart contract source code.

The audit process paid particular attention to the following considerations.

- A thorough review of the smart contract logic flow
- Assessment of the code base to ensure compliance with current best practice and industry standards
- Ensured the contract logic met the client's specifications and intent
- Internal vulnerability scanning tools tested for common risks and writing errors
- Testing smart contracts for common attack vectors
- Test smart contracts for known vulnerability risks
- Conduct a thorough line-by-line manual review of the entire code base

As a result of the security assessment, issues ranging from critical to informational were identified. We recommend that these issues are addressed to ensure a high level of security standards and industry practice. The recommendations we made could have better served the project from a security perspective.

- Enhance general coding practices to improve the structure of the source code.
- Provide more comments for each function to improve readability.
- Provide more transparency of privileged activities once the agreement is in place.

2 Overview

2.1 Project Summary

Project Summary	Project Information
Name	Acara
Website	https://acara.im/
Start date	July 7, 2023
End date	July 9, 2023
Platform	BNB Chain
Contract type	Token
Language	Solidity
File	Acara.sol

2.2 Audit Scope

File	SHA256
Acara.sol	EF8DFAC7F27ED0D02FA16546CE41CC00C8C419E6F55FE024F618028A46AF46FB

3 Project contract details

3.1 Contract Overview

Acara Contract

The contract uses the IBEP20 interface, uses the SafeMath security library, and uses the Context and Ownable contracts to determine the management privilege roles. The Acara contract mainly carries out the issuance of tokens, and there is no current contract with a total issuance of 30,000, and the contract has minting, destruction, and transfer functions.

3.2 Code Overview

Ownable Contract

Function Name	Visibility	Modifiers
owner	Public	-
renounceOwnership	Public	onlyOwner
transferOwnership	Public	onlyOwner
_transferOwnership	Internal	-

Acara Contract

Function Name	Visibility	Modifiers
getOwner	External	-
decimals	External	-
symbol	External	-
name	External	-
totalSupply	External	-
balanceOf	External	-
transfer	External	-
allowance	External	-
approve	External	-
transferFrom	External	-
increaseAllowance	Public	-
decreaseAllowance	Public	-
mint	Public	onlyOwner
burn	Public	-
_transfer	Internal	-
_mint	Internal	-
_burn	Internal	-
_approve	Internal	-
_burnFrom	Internal	-

4 Audit results

4.1 Key messages

ID	Title	Severity	Status
01	owner privileged role	Low	confirm

4.2 Audit details

4.2.1 owner privileged role

ID	Severity	Location	Status
01	Low	Acara.sol: 501, 504	confirm

Description

The mint() function is used to mint tokens. This method will continue to call the _mint() function to complete the minting, which can be called by the owner privileged role, which can perform the minting and may lead to security issues if the privileged role is maliciously controlled.

Code location:

```
493   /**
494    * @dev Creates `amount` tokens and assigns them to `msg.sender`, increasing
495    * the total supply.
496    *
497    * Requirements
498    *
499    * - `msg.sender` must be the token owner
500    */
501   function mint(uint256 amount) public onlyOwner returns (bool) {
502     _mint(_msgSender(), amount);
503     return true;
504   }
```

Recommendation

It is recommended that the privileged roles in the contract be managed using multiple signatures to avoid privileged roles being managed using EOA.

Status

confirm.

5 Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings 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 findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings 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 in-storage 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 but rather 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.

Disclaimer

This report is issued in response to facts that occurred or existed prior to the issuance of this report, and liability is assumed only on that basis.
Shield Security cannot determine the security status of this program and assumes no responsibility for facts occurring or existing after the date of this report. The security audit analysis and other content in this report is based on documents and materials provided to Shield Security by the information provider through the date of the insurance report. in Shield Security's opinion. The information provided is not missing, falsified, deleted or concealed. If the information provided is missing, altered, deleted, concealed or not in accordance with the actual circumstances, Shield Security shall not be liable for any loss or adverse effect resulting therefrom. shield Security will only carry out the agreed security audit of the security status of the project and issue this report. shield Security is not responsible for the background and other circumstances of the project. Shield Security is not responsible for the background and other circumstances of the project.