Audit report for BGACT Token

Prepared By: - Kishan Patel
Prepared On: - 12/12/2025.

Prepared for: Bluegrace Energy Bolivia

Table of contents

THIS AUDIT REPORT WILL CONTAIN CONFIDENTIAL INFORMATION ABOUT THE SMART CONTRACT AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES OF THEIR EXPLOITATION.
THE INFORMATION FROM THIS AUDIT REPORT CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.

Blockchain - Auditor Qualifications:

Mr. Kishan Patel - (kishanpatel412) - is a renowned and highly respected expert in the field of blockchain, tokenization and security crypto development, including areas of Regulatory Compliance, Privacy and general DeFi Security. He holds a B.Sc. bachelor's degree and also an M.Sc. degree in Applied Computer Science. He has performed well in excess of 1,000 smart contract reviews and audits.

1. 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). Because the total numbers of test cases are unlimited, 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.

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 their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

2. Introduction

Mr. Kishan Patel (Auditor & Consultant) was contacted by Bluegrace Energy Bolivia - BGEB. (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between 11/12/2025 – 12/12/2025.

The project has 1 file. It contains approx. 307 lines of Solidity code. All the functions and state variables are well commented on using the NATSPEC documentation, but that does not create any vulnerability.

3. Project information

Token Name	Bluegrace Amazon Carbon
Token Symbol	BGACT
Platform	Binance smart chain
Order Started Date	11/12/2025
Order Completed Date	12/12/2025

4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / The DAO hack
- Forcing BNB to a contract
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference

5. Severity Definitions

Risk	Level Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution

6. Good Coding Standards

- **Good, required condition in functions:-**
  - **Filename: ERC20.sol**
    - Here the smart contract checks that sender and recipient addresses are valid.

```
209     function _transfer(address sender, address recipient, uint256 amount
210         require(sender != address(0), "ERC20: transfer from the zero add
211         require(recipient != address(0), "ERC20: transfer to the zero ad
```

    - Here the smart contract is checking that account address is valid.

```
229     function _mint(address account, uint256 amount) internal virtual {
230         require(account != address(0), "ERC20: mint to the zero address"
231
```

    - Here the smart contract is checking that account address is valid.

```
250     function _burn(address account, uint256 amount) internal virtual {
251         require(account != address(0), "ERC20: burn from the zero addres
252
```

    - Here smart contract is checking that owner and spender addresses are valid.

```
273     function _approve(address owner, address spender, uint256 amount) in
274         require(owner != address(0), "ERC20: approve from the zero addre
275         require(spender != address(0), "ERC20: approve to the zero addre
```

  - **Filename: TeamToken.sol**
    - Here the smart contract is checking that owner, feeWallet addresses are valid, decimal is between 8 to 18. Supply is above 0.

```
19      constructor(
20          string memory name,
21          string memory symbol,
22          uint8 decimals,
23          uint256 supply,
24          address owner,
25          address feeWallet
26      ) public checkIsAddressValid(owner) checkIsAddressValid(feeWallet) E
27          require(decimals >=8 && decimals <= 18, "[Validation] Not valid
28          require(supply > 0, "[Validation] inital supply should be greate
```

7. Critical vulnerabilities in code

- **No Critical vulnerabilities found**

8. Medium vulnerabilities in code

- **No Medium vulnerabilities found**

9. Low vulnerabilities in code

9.1. Suggestions to add additional code validations:-

=> Have implemented the required validation in the contract.
=> There are potentially some basic additional improved validation and security of code to be made.
=> Note: These are all just suggestions and it is not related to any programming bug.

- **Function: - _approve**

```
273     function _approve(address owner, address spender, uint256 amount) in
274         require(owner != address(0), "ERC20: approve from the zero addre
275         require(spender != address(0), "ERC20: approve to the zero addre
```

- Here in _approve functions smart contract can check that owner has sufficient balance to provide allowance to sender.

10. Summary

- **Number of problems in the smart contract as per severity level**

Critical	Medium	Low
0	**0**	**1**

According to the assessment, the smart contract code is well secured. The code is written with all validation, and all security is implemented. Code is performing well and there is no way to steal funds from this contract.

- **Good Positive Point:** Code, performance and quality are good. All kinds of necessary validation are added into smart contract, and all validations are working as expected.

- **Suggestions & Comments:** Implement suggested code validations.

Addendum: Contract Self-Assessment & Technical Disclosure

This addendum is provided as a self-assessment for informational purposes only and does not replace an independent third-party security audit.

1. **Source Code Transparency**

The smart contract is open-source and published under the MIT license, as declared via the SPDX license identifier within the source code. The contract source code is publicly available and verifiable on BscScan.

2. **Bytecode Verification**

The deployed contract bytecode exactly matches the bytecode of a previously verified contract on BscScan:

0xe29046a93498b3676caac9a6c81911b5009bc0da

This confirms that the deployed contract executes the same compiled logic as the verified source code, with no modifications or hidden behavior introduced at deployment.

3. **Contract Design Characteristics**

Based on a manual review of the verified source code:

- The contract implements standard ERC-20 functionality using well-established libraries
- No proxy patterns, upgradeability mechanisms, or delegatecall usage are present
- No dynamic code execution or obfuscation techniques are used
- Any privileged or owner-restricted functions are explicitly defined and visible in the source code

4. **Risk Scope Statement**

This self-assessment confirms code transparency and structural simplicity but does not guarantee the absence of vulnerabilities under all conditions, nor does it replace a professional third-party security audit.

