auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

BGIGGLE

DATED : 6 November 2025

AUDIT SUMMARY

Project name – BGIGGLE

Date: 6 November 2025

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: **Passed with high risk**

Issues Found

Status	Critical	High	Medium	Low	Informational
Open	0	1	0	0	1
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it.

3- Slither :
The code has undergone static analysis using Slither.

Token Information

Token Address:
0x68aDE45010D3Da4Bf9aBD6ea57186ca9C506F6a9

Name: Baby Giggle

Symbol: BGIGGLE

Decimals: 9

Network: BscScan

Type: BSC

Owner: 0x814EB97bAa03FDB76fE8B16D676605B212bf036a

Deployer:
0x814EB97bAa03FDB76fE8B16D676605B212bf036a

Token Supply: 1,000,000

Checksum: Je032c616934aeb47e6039f76b20d613

TOKEN OVERVIEW

Buy Fee: 0-5%

Sell Fee: 0-5%

Fee Privilege: Owner

Ownership: Owned

Minting: None

Max Tx: No

Blacklist: No

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- Return values of low-level calls
- Private modifier
- Multiple Sends
- Using Suicide
- Gas Limitand Loops
- Address hardcoded
- Exception Disorder
- Using inline assembly
- Divide before multiply
- Missing Zero Address Validation
- Compiler version not fixed
- **Gasless Send**
- Using block.timestamp
- Re-entrancy
- Tautology or contradiction
- Timestamp Dependence
- Revert/require functions
- Use of tx.origin
- Integer overflow/underflow
- Dangerous strict equalities
- Using SHA3
- Using throw

INHERITANCE TREE

POINTS TO NOTE

- The owner can setExemptFrom Fee.
- The owner can update updateSwapTokensAmt.
- The owner can Enable trading.
- The owner can update taxes.

CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	0
High-Risk	1
Medium-Risk	0
Low-Risk	0
Optimization/ Informational	1

MANUAL TESTING

Centralization –Enabling Trades

Severity: High

Function: EnableTrading

Status: Open

Overview:

The EnableTrading function permits only the contract owner to activate trading capabilities. Until this function is executed, no investors can buy, sell, or transfer their tokens. This places a high degree of control and centralization in the hands of the contract owner.

```
function enableTrading() external onlyOwner {
    require(!tradingAllowed, "Trading already enabled");
    tradingAllowed = true;
    launchBlock = block.number;
    lastSwapBackBlock = block.number;
  }
```

Suggestion:

To reduce centralization and potential manipulation, consider one of the following approaches:

1. Automatically enable trading after a specified condition, such as the completion of a presale, is met.
2. If manual activation is still desired, consider transferring the ownership of the contract to a trustworthy, third-party entity like a certified "PinkSale Safu" developer. This can give investors more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad-faith actions by the original owner.

MANUAL TESTING

Optimization

Subject: Remove unused code

Severity: Optimization

Status: Open

Overview:
Unused variables are allowed in Solidity, and they do. not pose a direct security issue. It is the best practice. though to avoid them.

```
function sendValue(address payable recipient, uint256 amount)
internal {
   require(
    address(this).balance >= amount,
    "Address: insufficient balance"
   );

   (bool success, ) = recipient.call{value: amount}("");
   require(
    success,
    "Address: unable to send value, recipient may have reverted"
   );
 }

 function functionCall(address target, bytes memory data)
  internal
  returns (bytes memory)
 {
  return
   functionCallWithValue(
    target,
    data,
    0,
    "Address: low-level call failed"
```

MANUAL TESTING

```
        );
    }
    function functionCallWithValue(       address target,
        bytes memory data,
        uint256 value
    ) internal returns (bytes memory) {
        return
            functionCallWithValue(
                target,
                data,
                value,
                "Address: low-level call with value failed"
            );
    }
function functionStaticCall(address target, bytes memory data)
        internal
        view
        returns (bytes memory)
    {
        return
            functionStaticCall(
                target,
                data,
                "Address: low-level static call failed"
            );
    }
```

MANUAL TESTING

```
function functionDelegateCall(address target, bytes memory data)
    internal
    returns (bytes memory)
  {
    return
      functionDelegateCall(
        target,
        data,
        "Address: low-level delegate call failed"
      );
  }

library SafeERC20 { //@audit unsed library
  using Address for address;


function safeTransfer(
    IERC20 token,
    address to,
    uint256 value
  ) internal {
    _callOptionalReturn(
      token,
      abi.encodeWithSelector(token.transfer.selector, to, value)
    );
  }
```

MANUAL TESTING

```
function functionDelegateCall(address target, bytes memory data)
    internal
    returns (bytes memory)
  {
    return
      functionDelegateCall(
        target,
        data,
        "Address: low-level delegate call failed"
      );
  }

library SafeERC20 {
  using Address for address;

  function safeTransfer(
    IERC20 token,
    address to,
    uint256 value
  ) internal {
    _callOptionalReturn(
      token,
      abi.encodeWithSelector(token.transfer.selector, to, value)
    );
  }

  function safeTransferFrom(
  IERC20 token,
  address from,
  address to,
  uint256 value
  ) internal {
  _callOptionalReturn(
  token,
  abi.encodeWithSelector(token.transferFrom.selector, from, to, value)
  );
  }
```

MANUAL TESTING

```
 function safeApprove(
    IERC20 token,
    address spender,
    uint256 value
  ) internal {
    // safeApprove should only be called when setting an initial
allowance,
    // or when resetting it to zero. To increase and decrease it, use
    // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
    require(
      (value == 0) || (token.allowance(address(this), spender) == 0),
      "SafeERC20: approve from non-zero to non-zero allowance"
    );
    _callOptionalReturn(
      token,
      abi.encodeWithSelector(token.approve.selector, spender, value)
    );
  }
}

interface ILpPair {
  function sync() external;
}


 event UpdatedTransactionLimit(uint256 newMax);
  event UpdatedWalletLimit(uint256 newMax);
  event SetExemptFromFees(address _address, bool _isExempt);
  event SetExemptFromLimits(address _address, bool _isExempt);
```

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace