Coinsult

Advanced Manual Smart Contract Audit

Project: Baby frog inu

Website: https://www.thebabyfroginu.com/

Low-Risk	Medium-Risk	High-Risk
7 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

0xfE29FF7b04e6755bCc8d17988F16F81C7186cbcD

Disclaimer: Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

Disclaimer

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x60be8cd915f9e7d998dd14da5a551168dbcb248d	54,700,000,000	54.7000%
2	0x68f601ddc586f7cb48919397888b0b708876f411	45,300,000,000	45.3000%

Source Code

Coinsult was comissioned by Baby frog inu to perform an audit based on the following smart contract:

https://bscscan.com/address/0xfe29ff7b04e6755bcc8d17988f16f81c7186cbcd#code

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

7 low-risk code
issues found

Medium-Risk

0 medium-risk code
issues found

High-Risk

0 high-risk code
issues found

The detailed report continues on the next page...

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Contract contains Reentrancy vulnerabilities

Additional information: This combination increases risk of malicious intent. While it may be justified by some complex mechanics (e.g. rebase, reflections, buyback).

More information: Slither

```
function _transfer(address from, address to, uint256 amount) private {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount &gt; 0, "Transfer amount must be greater than zero");

    _redisFee = 0;
    _taxFee = 0;

    if (from != owner() &amp;&amp; to != owner()) {

        uint256 contractTokenBalance = balanceOf(address(this));
        if (!inSwap &amp;&amp; from != uniswapV2Pair &amp;&amp; swapEnabled &amp;&amp; contractTokenI
            swapTokensForEth(contractTokenBalance);
            uint256 contractETHBalance = address(this).balance;
            if(contractETHBalance &gt; 0) {
                sendETHToFee(address(this).balance);
            }
        }

        if(from == uniswapV2Pair &amp;&amp; to != address(uniswapV2Router)) {
            _redisFee = _redisFeeOnBuy;
            _taxFee =  _taxFeeOnBuy;
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if mgs.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router), tokenAmount);
    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

Recommendation

Do not use `block.timestamp`, now or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
      reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 private constant _tTotal = 100000 * 10**6 * 10**9;
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

While `1_ether` looks like `1 ether`, it is `10 ether`. As a result, it's likely to be used incorrectly.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

No zero address validation for some functions

Detect missing zero address validation.

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

Recommendation

Check that the new address is not zero.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

Bob calls `updateOwner` without specifying the `newOwner`, soBob loses ownership of the contract.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Functions that send Ether to arbitrary destinations

Unprotected call to a function sending Ether to an arbitrary address.

```
function sendETHToFee(uint256 amount) private {
    _developmentAddress.transfer(amount.div(2));
    _marketingAddress.transfer(amount.div(2));
}
```

Recommendation

Ensure that an arbitrary user cannot withdraw unauthorized funds.

Exploit scenario

```
contract ArbitrarySend{
    address destination;
    function setDestination(){
        destination = msg.sender;
    }

    function withdraw() public{
        destination.transfer(this.balance);
    }
}
```

Bob calls `setDestination` and `withdraw`. As a result he withdraws the contract's balance.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
function rescueForeignTokens(address _tokenAddr, address _to, uint _amount) public onlyDev() {
    emit tokensRescued(_tokenAddr, _to, _amount);
    Token(_tokenAddr).transfer(_to, _amount);
}
```

Recommendation

Use `SafeERC20`, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in `MyBank`, `deposit` will not revert if the transfer fails, and an attacker can call `deposit` for free..

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Conformance to Solidity naming conventions

Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
string private constant _name = "Baby Frog Inu";
string private constant _symbol = "BFGI";
uint8 private constant _decimals = 9;
```

Recommendation

Follow the Solidity naming convention.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (`ERC20`).
- Allow _ at the beginning of the `mixed_case` match for private variables and unused parameters.

Owner privileges

- Owner cannot set fees higher than 25%
- Owner cannot pause trading
- Owner cannot change max transaction amount
- Owner can exclude from fees

Extra notes by the team

No notes

Contract Snapshot

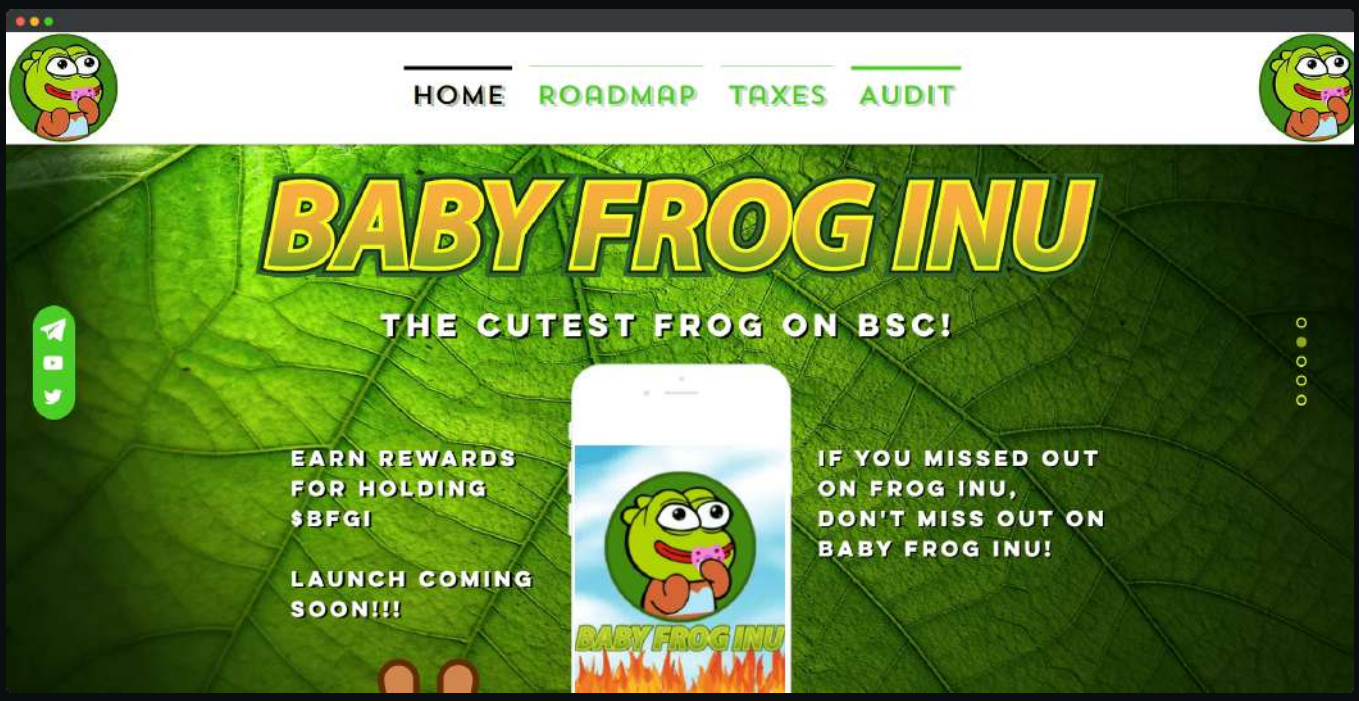
```
contract BabyFrogInu is Context, IERC20, Ownable {

using SafeMath for uint256;
mapping (address =&gt; uint256) private _rOwned;
mapping (address =&gt; uint256) private _tOwned;
mapping (address =&gt; mapping (address =&gt; uint256)) private _allowances;
mapping (address =&gt; bool) private _isExcludedFromFee;

uint256 private constant MAX = ~uint256(0);
uint256 private constant _tTotal = 100000 * 10**6 * 10**9;
uint256 private _rTotal = (MAX - (MAX % _tTotal));
uint256 private _tFeeTotal;
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



- Mobile Friendly
- Does not contain jQuery errors
- SSL Secured
- No major spelling errors

Project Overview

Not KYC verified by Coinsult

Baby frog inu

Audited by Coinsult.net

Date: 11 June 2022

Advanced Manual Smart Contract Audit