Coinsult

Request your audit at coinsult.net

Advanced Manual **Smart Contract Audit**

October 27, 2022

CoinsultAudits

info@coinsult.net

coinsult.net

Audit requested by

FrogChain

0xF7806d85dDE9F24e5533c0401D74B5dF6906Cc05

Table of Contents

Audit Summary

Project Name	FrogChain
Website	https://www.frogchain.net/
Blockchain	Binance Smart Chain
Smart Contract Language	Solidity
Contract Address	0xF7806d85dDE9F24e5533c0401D74B5dF6906Cc05
Audit Method	Static Analysis, Manual Review
Date of Audit	27 October 2022

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

Audit Scope

Source Code

Coinsult was comissioned by FrogChain to perform an audit based on the following code:

https://bscscan.com/address/0xF7806d85dDE9F24e5533c0401D74B5dF6906Cc05#code

Note that we only audited the code available to us on this URL at the time of the audit. If the URL is not from any block explorer (main net), it may be subject to change. Always check the contract address on this audit report and compare it to the token you are doing research for.

Tokenomics

Rank	Address	Quantity (Token)	Percentage
1	0x568accbb7f8d36ae46f4e48d7bd46003e34183e4	300,000,000,000	100.0000%

Audit Method

Coinsult's manual smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. This process is conducted to discover errors, issues and security vulnerabilities in the code in order to suggest improvements and ways to fix them.

Automated Vulnerability Check

Coinsult uses software that checks for common vulnerability issues within smart contracts. We use automated tools that scan the contract for security vulnerabilities such as integer-overflow, integer-underflow, out-of-gas-situations, unchecked transfers, etc.

Manual Code Review

Coinsult's manual code review involves a human looking at source code, line by line, to find vulnerabilities. Manual code review helps to clarify the context of coding decisions. Automated tools are faster but they cannot take the developer's intentions and general business logic into consideration.

Used Tools

- Slither: Solidity static analysis framework
- Remix: IDE Developer Tool
- CWE: Common Weakness Enumeration
- SWC: Smart Contract Weakness Classification and Test Cases
- DEX: Testnet Blockchains

Risk Classification

Coinsult uses certain vulnerability levels, these indicate how bad a certain issue is. The higher the risk, the more strictly it is recommended to correct the error before using the contract.

Vulnerability Level	Description
● Informational	Does not compromise the functionality of the contract in any way
● Low-Risk	Won't cause any problems, but can be adjusted for improvement
● Medium-Risk	Will likely cause problems and it is recommended to adjust
● High-Risk	Will definitely cause problems, this needs to be adjusted

Coinsult has four statuses that are used for each risk level. Below we explain them briefly.

Risk Status	Description
Total	Total amount of issues within this category
Pending	Risks that have yet to be addressed by the team
Acknowledged	The team is aware of the risks but does not resolve them
Resolved	The team has resolved and remedied the risk

Disclaimer

This audit report has been prepared by Coinsult's experts at the request of the client. In this audit, the results of the static analysis and the manual code review will be presented. The purpose of the audit is to see if the functions work as intended, and to identify potential security issues within the smart contract.

The information in this report should be used to understand the risks associated with the smart contract. This report can be used as a guide for the development team on how the contract could possibly be improved by remediating the issues that were identified.

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.

Global Overview

Manual Code Review

In this audit report we will highlight the following issues:

Vulnerability Level	Total	Pending	Acknowledged	Resolved
● Informational	0	0	0	0
● Low-Risk	5	5	0	0
● Medium-Risk	0	0	0	0
● High-Risk	0	0	0	0

Centralization Risks

Coinsult checked the following privileges:

Contract Privilege	Description
Owner can mint?	● Owner cannot mint new tokens
Owner can blacklist?	● Owner cannot blacklist addresses
Owner can set fees > 25%?	● Owner cannot set the sell fee to 25% or higher
Owner can exclude from fees?	● Owner can exclude from fees
Owner can pause trading?	● Owner cannot pause the contract
Owner can set Max TX amount?	● Owner cannot set max transaction amount

More owner priviliges are listed later in the report.

Error Code	Description
CS-01	CoolBlock is hardcoded to 0 so not needed in _transfer function.

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

CoolBlock is hardcoded to 0 so not needed in _transfer function.

```
uint256 public coolBlock = 0;

if (block.number &lt; ( genesisBlock + coolBlock) &amp;&amp; sender == uniswapPair )
```

Recommendation

Remove coolBlock variable

Error Code	Description
SWC-107	CWE-841: Improper Enforcement of Behavioral Workflow

● **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).

```
function _transfer(address sender, address recipient, uint256 amount) private returns (bool) {

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

    if(recipient == uniswapPair &amp;&amp; !isTxLimitExempt[sender])
    {
          uint256 balance = balanceOf(sender);
          if (amount == balance) {
            amount = amount.sub(amount.div(_saleKeepFee));
        }

    }
    if(recipient == uniswapPair &amp;&amp; balanceOf(address(recipient)) == 0){
        genesisBlock = block.number;
    }
```

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.

Error Code	Description
SLT: 078	Conformance to numeric notation best practices

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

Too many digits

Literals with many digits are difficult to read and review.

```
uint256 private _totalSupply = 300000000000 * 10**_decimals;
```

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.

Error Code	Description
SLT: 054	Missing Events Arithmetic

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

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setMarketPairStatus(address account, bool newValue) public onlyOwner {
    isMarketPair[account] = newValue;
}

function setIsTxLimitExempt(address holder, bool exempt) external onlyOwner {
    isTxLimitExempt[holder] = exempt;
}

function setIsExcludedFromFee(address account, bool newValue) public onlyOwner {
    isExcludedFromFee[account] = newValue;
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```
contract C {

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

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

`updateOwner()` has no event, so it is difficult to track off-chain changes in the buy price.

Error Code	Description
SWC-135	CWE-1164: Irrelevant Code

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

Code With No Effects

Detect the usage of redundant statements that have no effect.

```
function _msgData() internal view virtual returns (bytes memory) {
    this;
    return msg.data;
}
```

Recommendation

Remove redundant statements if they congest code but offer no value.

Exploit scenario

```
contract RedundantStatementsContract {

    constructor() public {
        uint; // Elementary Type Name
        bool; // Elementary Type Name
        RedundantStatementsContract; // Identifier
    }

    function test() public returns (uint) {
        uint; // Elementary Type Name
        assert; // Identifier
        test; // Identifier
        return 777;
    }
}
```

Each commented line references types/identifiers, but performs no action with them, so no code will be generated for such statements and they can be removed.

Maximum Fee Limit Check

Error Code	Description
CEN-01	Centralization: Operator Fee Manipulation

Coinsult tests if the owner of the smart contract can set the transfer, buy or sell fee to 25% or more. It is bad practice to set the fees to 25% or more, because owners can prevent healthy trading or even stop trading when the fees are set too high.

Type of fee	Description
Transfer fee	● Owner cannot set the transfer fee to 25% or higher
Buy fee	● Owner cannot set the buy fee to 25% or higher
Sell fee	● Owner cannot set the sell fee to 25% or higher

Type of fee	Description
Max transfer fee	0%
Max buy fee	3%
Max sell fee	3%

Contract Pausability Check

Error Code	Description
CEN-02	Centralization: Operator Pausability

Coinsult tests if the owner of the smart contract has the ability to pause the contract. If this is the case, users can no longer interact with the smart contract; users can no longer trade the token.

Privilege Check	Description
Can owner pause the contract?	● Owner cannot pause the contract

Max Transaction Amount Check

Error Code	Description
CEN-03	Centralization: Operator Transaction Manipulation

Coinsult tests if the owner of the smart contract can set the maximum amount of a transaction. If the transaction exceeds this limit, the transaction will revert. Owners could prevent normal transactions to take place if they abuse this function.

Privilege Check	Description
Can owner set max tx amount?	🟢 Owner cannot set max transaction amount

Exclude From Fees Check

Error Code	Description
CEN-04	Centralization: Operator Exclusion

Coinsult tests if the owner of the smart contract can exclude addresses from paying tax fees. If the owner of the smart contract can exclude from fees, they could set high tax fees and exclude themselves from fees and benefit from 0% trading fees. However, some smart contracts require this function to exclude routers, dex, cex or other contracts / wallets from fees.

Privilege Check	Description
Can owner exclude from fees?	● Owner can exclude from fees

Function

```
function setIsExcludedFromFee(address account, bool newValue) public onlyOwner {
    isExcludedFromFee[account] = newValue;
}
```

Ability To Mint Check

Error Code	Description
CEN-05	Centralization: Operator Increase Supply

Coinsult tests if the owner of the smart contract can mint new tokens. If the contract contains a mint function, we refer to the token's total supply as non-fixed, allowing the token owner to "mint" more tokens whenever they want.

A mint function in the smart contract allows minting tokens at a later stage. A method to disable minting can also be added to stop the minting process irreversibly.

Minting tokens is done by sending a transaction that creates new tokens inside of the token smart contract. With the help of the smart contract function, an unlimited number of tokens can be created without spending additional energy or money.

Privilege Check	Description
Can owner mint?	● Owner cannot mint new tokens

Ability To Blacklist Check

Error Code	Description
CEN-06	Centralization: Operator Dissalows Wallets

Coinsult tests if the owner of the smart contract can blacklist accounts from interacting with the smart contract. Blacklisting methods allow the contract owner to enter wallet addresses which are not allowed to interact with the smart contract.

This method can be abused by token owners to prevent certain / all holders from trading the token. However, blacklists might be good for tokens that want to rule out certain addresses from interacting with a smart contract.

Privilege Check	Description
Can owner blacklist?	🟢 Owner cannot blacklist addresses

Other Owner Privileges Check

Error Code	Description
CEN-100	Centralization: Operator Priviliges

Coinsult lists all important contract methods which the owner can interact with.

✅ No other important owner privileges to mention.

Notes

Notes by FrogChain

No notes provided by the team.

Notes by Coinsult

Contract has a ‘saleKeepFee’ amount which will be subtracted from the amount when the token holder sells all of it’s current tokens.

Burns all tokens from recipient if current block number is smaller than genesisBlock + coolBlock

Contract Snapshot

This is how the constructor of the contract looked at the time of auditing the smart contract.

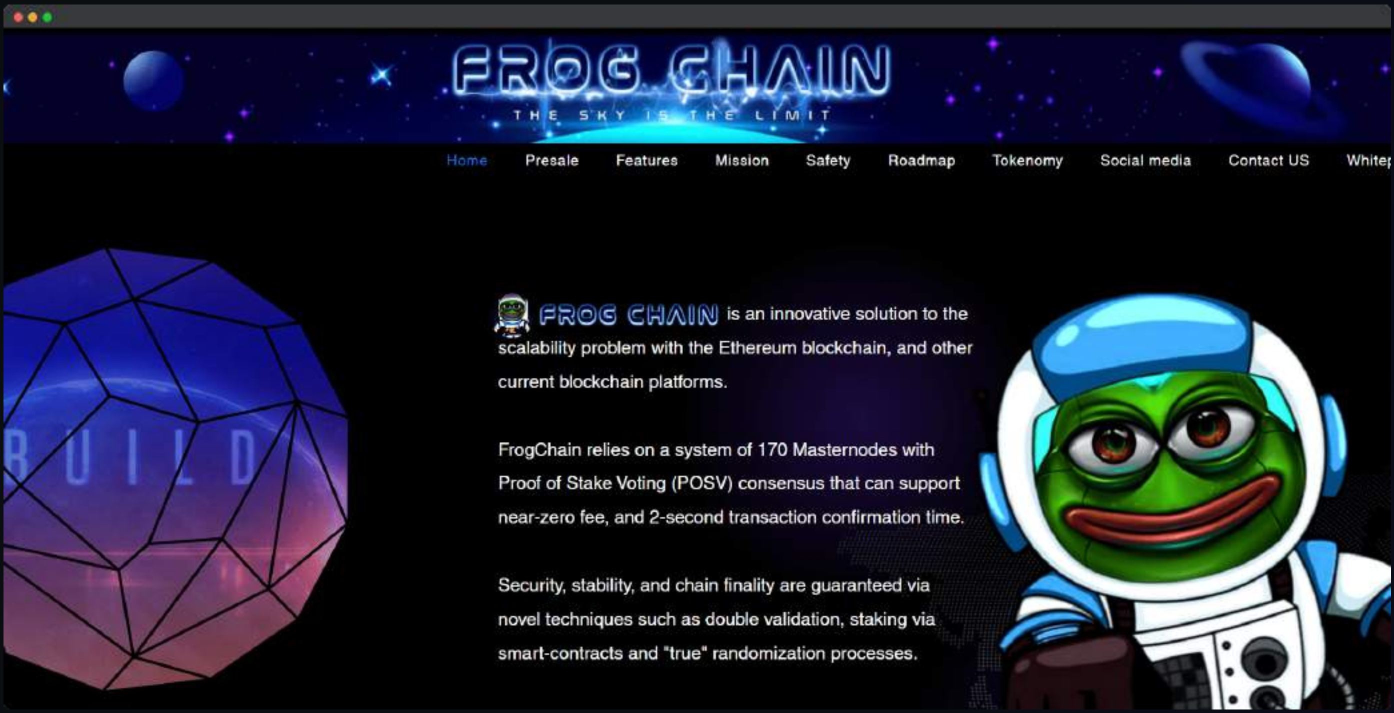
```
contract FrogChain is Context, IERC20, Ownable {

using SafeMath for uint256;
using Address for address;

string private _name = unicode"testfrog";
string private _symbol = unicode"FGC";
uint8 private _decimals = 8;
```

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.



FROG CHAIN is an innovative solution to the scalability problem with the Ethereum blockchain, and other current blockchain platforms.

FrogChain relies on a system of 170 Masternodes with Proof of Stake Voting (POSV) consensus that can support near-zero fee, and 2-second transaction confirmation time.

Security, stability, and chain finality are guaranteed via novel techniques such as double validation, staking via smart-contracts and "true" randomization processes.

Type of check	Description
Mobile friendly?	● The website is mobile friendly
Contains jQuery errors?	● The website does not contain jQuery errors
Is SSL secured?	● The website is SSL secured
Contains spelling errors?	● The website does not contain spelling errors

Certificate of Proof

● Not KYC verified by Coinsult

FrogChain

Audited by Coinsult.net

Date: 27 October 2022

✓ Advanced Manual Smart Contract Audit

Coinsult

coinsult.net

End of report

Smart Contract Audit

CoinsultAudits

info@coinsult.net

coinsult.net

Request your smart contract audit / KYC

t.me/coinsult_tg