Security Audit Report

PepeSaga ($PESA)

BEP20 on Binance Smart Chain

11st of May, 2023

https://github.com/ProteX-Audit

Table of Contents

Overview

Project Summary	
Project Name	PepeSAGA ($PESA)
Platform	Binance Smart Chain
Language	Solidity
Contract Type	BEP20
Contract Address	0x3A21E709aad8a06fe01d96109bEe2b781f1d
Block Explorer	https://bscscan.com/

Audit Summary	
Delivery Date	**May:** 11st, 2023 GMT+0
Block Number	21872310
Static Analysis	Yes
Graphic Analysis	Yes
Logic Disassemble	Yes
Mannual Review	Yes

Vulnerability Summary

Severity Level	Total	Acknowledged	Alleviated	Resolved
Critical	0	0	0	0
Major	0	0	0	0
Medium	0	0	0	0
Minor	0	0	0	0
Informational	0	0	0	0
Discussion	1	1	0	0

Fully Sanity Checks

	Read	Write	AI Scanned	Human Reviewed	Result	Suggested	Resolved
name()	Yes		Completed	Completed	No Risk		
symbol()	Yes		Completed	Completed	No Risk		
balanceOf()	Yes		Completed	Completed	No Risk		
decimals()	Yes		Completed	Completed	No Risk		
totalSupply()	Yes		Completed	Completed	No Risk		
allowance()	Yes		Completed	Completed	No Risk		
approve()		Yes	Completed	Completed	No Risk		
burn()		Yes	Completed	Completed	No Risk		
decreaseAllowance()		Yes	Completed	Completed	✓ Low/No Risk		
increaseAllowance()		Yes	Completed	Completed	✓ Low/No Risk		
renounceOwnership()		Yes	Completed	Completed	✓ Low/No Risk		
transfer()		Yes	Completed	Completed	✓ Low/No Risk		
transferFrom()		Yes	Completed	Completed	✓ Low/No Risk		
transferOwnership()		Yes	Completed	Completed	✓ Low/No Risk		

Source Code Analysis

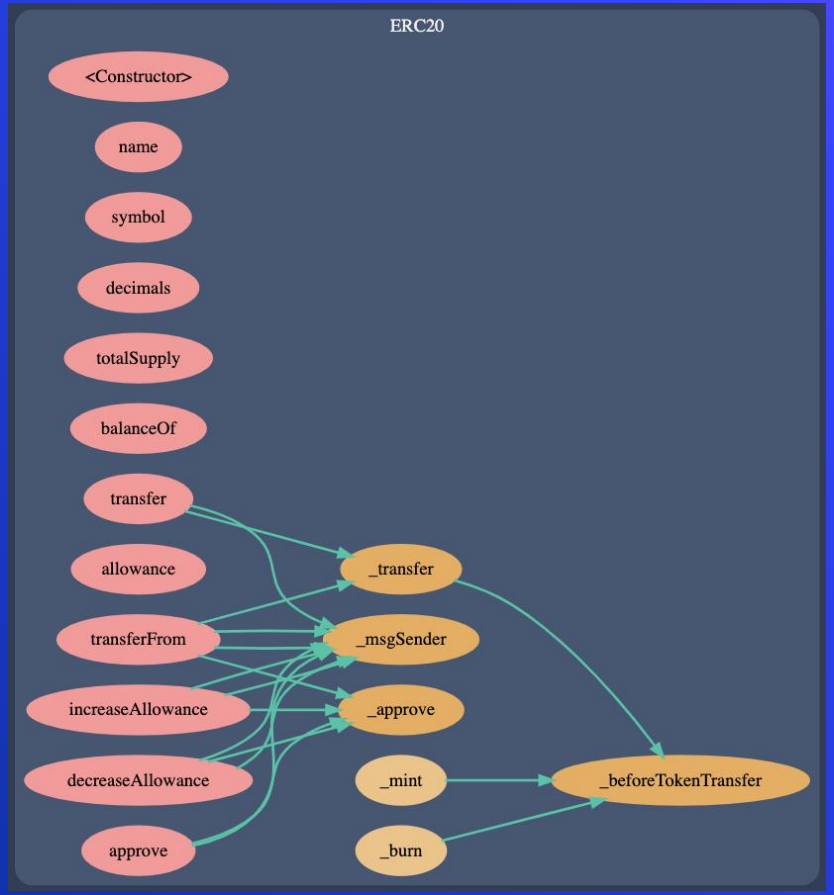
```
contract ERC20 is Context, IERC20 {
    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    constructor (string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    function name() public view virtual returns (string memory) {
        return _name;
    }

    function symbol() public view virtual returns (string memory) {
        return _symbol;
    }

    function decimals() public view virtual returns (uint8) {
        return 18;
    }

    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view virtual override returns (uint256) {
        return _balances[account];
    }

    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);

        uint256 currentAllowance = _allowances[sender][_msgSender()];
        require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
        _approve(sender, _msgSender(), currentAllowance - amount);

        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender] + addedValue);
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        uint256 currentAllowance = _allowances[_msgSender()][spender];
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
        _approve(_msgSender(), spender, currentAllowance - subtractedValue);

        return true;
    }

    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        uint256 senderBalance = _balances[sender];
        require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
        _balances[sender] = senderBalance - amount;
        _balances[recipient] += amount;

        emit Transfer(sender, recipient, amount);
    }
```

We've found 6 contracts in $PESA project source code and the partial screenshot of the contract code as left side shown.

- $PESA,
- Ownable,
- ERC20,
- Pausable,
- Context,
- IERC20,

respectively.

And $PESA is an interface in which will be implementing in later time.



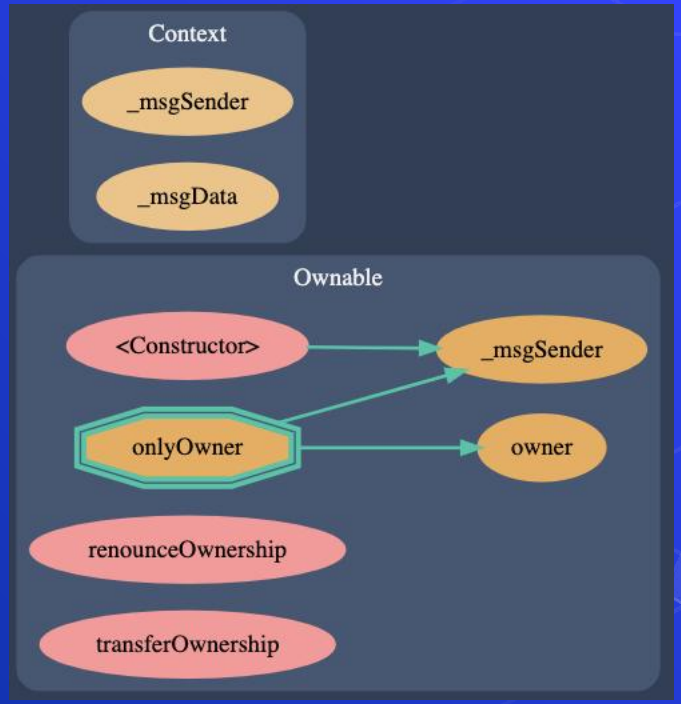
ERC20 Contract

- name()
- symbol()
- decimals()
- totalSupply()
- balanceOf()
- allowance()

Read functions are running as expected while analyzing at the time of this writing.

- transferFrom()
- transfer()
- increaseAllowance()
- decreaseAllowance()
- approve()

Write functions are in no risk at the time of this writing



Context Contract

No public or external function can be called

Ownable Contract

- renounceOwnership()
- transferOwnership()

The owner of the contract can initiate ownership renounced or transferred.

Contract Ownership

Contract Ownership Has Not Been Renounced at the Time of Audit.

The contract ownership is not currently renounced. It's because the project team will implement the game contract and set the implementation solidity contract in later time.

We just placed the contract of the owner address below for you to look up: 0x8488f2ce4423d5f4ac5742f1a9a9e64e8539f5ac

Some feasible suggestions that would also mitigate the potential risk at a different level for priviledged ownership.

- Time-lock with reasonable latency, e.g., 48 hours for awareness on priviledged operations
- Assignment of priviledged roles to multi-signature wallets to prevent a single point of failure, for example, due to the private key compromised

Liquidity Ownership

No Lock/Unlock Liquidity Logic Code Has Been Found.

This page will contain if there is that code that links to locked liquidity for the project if we are able to locate that information.

Locked liquidity information was neither found on the project's website nor inside the contracts.

Mint Function

The Contract Cannot Mint New $PESA Tokens.

We do understand that Mint functions are crucial to the functionality of the project, it's core related to its investors.

But a mint function was not found in the contract code.

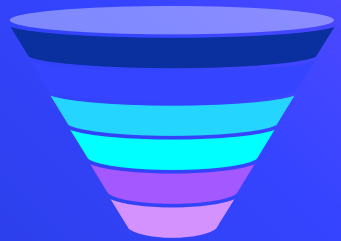
Burn Function

The Contract Has a Burn Function.

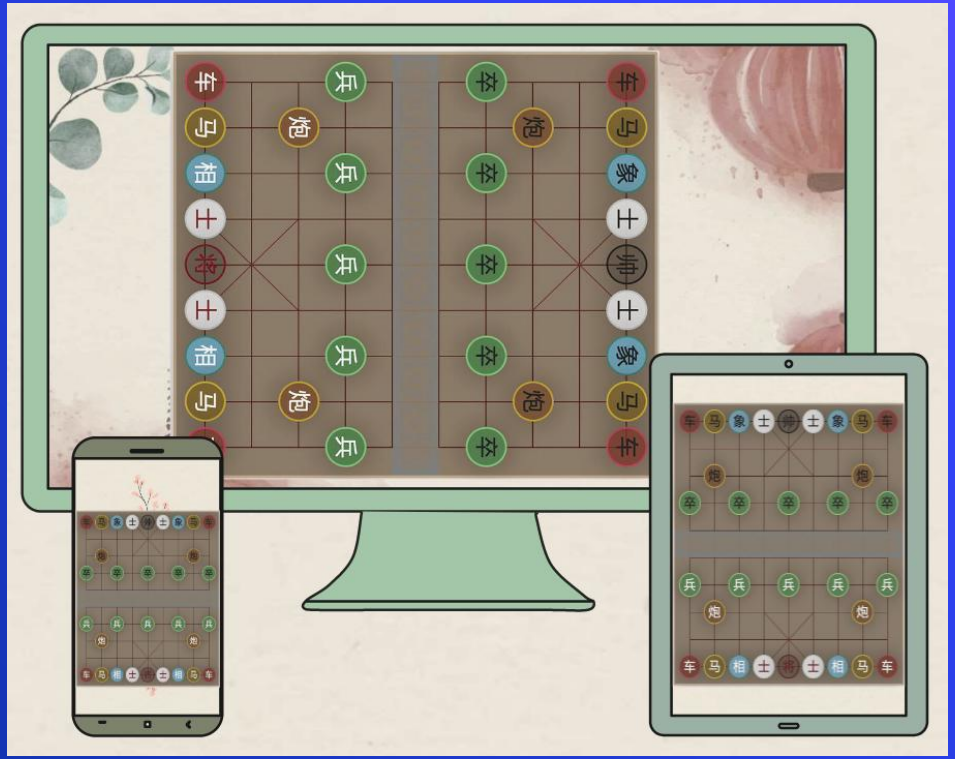
```
34      function burn(uint256 amount) public onlyOwner {
35          _burn(_msgSender(), amount);
36      }
37
```

The Burn function works well as expected according to buy back from the market periodically.

A burn function was found in the contract code.



Present Mode



The left image is an actual snapshot of the current live website.

The website was registered on May-08-2023.

Team Location

PepeSAGA Official Website: https://pepesaga.com/

General Web Security

DOMAIN

A valid domain hosted by Cloudflare.

Registered on 08-May-2023

https://pepesaga.com/

Social Media Accounts

A bundle of social media accounts was found.

Twitter: https://twitter.com/pesa_pepesaga

Telegram: https://t.me/PepeSAGA_Official_Group

A legal SSL certificate was found.

Issued at 08- May-2023

Signature Algorithm is sha256WithRSAEncryption

SSL CERTIFICATE

No malware found.
No injected spam found.
No internal server errors.

Domain is marked clean by Google and McAfee.

SPAM/MALWARE

Disclaimer

The opinions expressed in this document are for general informational purposes only and **are not intended to provide specific advice or recommendations for any individual or on any specific investment**. It is only intended to provide education and public knowledge regarding to this projects. This audit is only applied to the type of auditing specified in this report and the scope of given in the results. Other unknown security vulnerabilities are beyond responsibility. ProteXAudit only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, ProteXAudit lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The smart contract analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to ProteXAudit or was publicly available before the issuance of this report (issuance of report recorded via block number on cover page), if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, ProteXAudit assumes no responsibility for the resulting loss or adverse effects. Due to the technical limitations of any organization, this report conducted by PROTEX still has the possibility that the entire risk cannot be completely detected. ProteXAudit disclaims any liability for the resulting losses.

ProteXAudit provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Even projects with a low risk score have been known to pull liquidity, sell all team tokens, or exit scam. Please exercise caution when dealing with any cryptocurrency related platforms. The final interpretation of this statement belongs to ProteXAudit.

ProteXAudit highly advises against using cryptocurrencies as speculative investments and they should be used solely for the utility they aim to provide.

About

ProteX Audit has founded in 2021 by a squad of elite geeks on blockchain research and we analyze the loopholes in most smart contracts in Binance and ethereum-based chains. We offer the best-in-class report for your smart contracts auditing. Customers trust smart contract assessment report, investors trust the project.

https://github.com/ProteX-Audit

support@protexaudit.tech