BLOCK SOLUTIONS

Smart Contract Code Review and Security Analysis Report for PURRPAWS BEP20 Token Smart Contract

Request Date: 2025-01-26
Completion Date: 2025-01-27
Language: Solidity

Contents

Commission

Audited Project	PURRPAWS BEP20 Token Smart Contract
Smart Contract Address	0x0De62a081059F914E20b9175314E8AC5688F5D61
Contract Creator	0xA8539536066eF8B58840819c19079e1D16461d7A
Contract Owner	0xA8539536066eF8B58840819c19079e1D16461d7A
Blockchain Network	Binance Smart Chain Mainnet

Block Solutions was commissioned by PURRPAWS BEP20 Token Smart Contract owners to perform an audit of their main smart contract. The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

PURRPAWS Properties

Contract Token name	PURRPAWS
Total supply	100,000,000,000 PPAWS
Symbol	PPAWS
Decimals	18
Max Wallet Holdings	100000000000 PPAWS
Max Transaction Value	100000000000 PPAWS
Holders	8
Total Transfers	9
Buy Fee	5%
Sell Fee	5 %
Transfer Fee	0%
Base Token	0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
Liquidity Wallet	0x576AF81FD4Ff765cB5a199b14109eCE191e1db7e
Marketing Wallet	0x6bA9dd0416a2943cb8618e6686BB5b0d0878925f
Uniswap v2 Pair	0x7eD9B0b18bc2f6C3DF386296D81e97DFa1f2A4A2
Uniswap V2 Router	0x10ED43C718714eb63d5aA57B78B54704E256024E
Smart Contract Address	0x0De62a081059F914E20b9175314E8AC5688F5D61
Contract Creator	0xA8539536066eF8B58840819c19079e1D16461d7A
Contract Owner	0xA8539536066eF8B58840819c19079e1D16461d7A
Blockchain Network	Binance Smart Chain Mainnet

Contract Functions

Executables

i. function approve(address spender, uint256 value) public virtual returns (bool)
ii. function claimLeftoverBNB() external onlyOwner
iii. function mint(address recipient, uint256 amount) external onlyOwner
iv. function renounceOwnership() public virtual onlyOwner
v. function transferOwnership(address newOwner) public virtual onlyOwner
vi. function setLiquidityWallet(address _liquidityWallet) external onlyOwner
vii. function setMarketingWallet(address _marketingWallet) external onlyOwner
viii. function setMaxWalletAndTxAmount(uint256 _maxWalletAmount, uint256 _maxTxAmount) external onlyOwner
ix. function setTaxExemption(address account, bool exempt) external onlyOwner
x. function setTaxes(uint256 _buyTaxMarketing,uint256 _buyTaxLiquidity,uint256 _sellTaxMarketing,uint256 _sellTaxLiquidity) external onlyOwner
xi. function setThreshold(uint256 _threshold) external onlyOwner
xii. function swapAndDistributeBNB() public
xiii. function transfer(address to, uint256 value) public virtual returns (bool)
xiv. function transferFrom(address from, address to, uint256 value) public virtual returns (bool)

Checklist

Compiler errors.	Passed
Possible delays in data delivery.	Passed
Timestamp dependence.	Passed
Integer Overflow and Underflow.	Passed
Race Conditions and Reentrancy.	Passed
DoS with Revert.	Passed
DoS with block gas limit.	Passed
Methods execution permissions.	Passed
Economy model of the contract.	Passed
Private user data leaks.	Passed
Malicious Events Log.	Passed
Scoping and Declarations.	Passed
Uninitialized storage pointers.	Passed
Arithmetic accuracy.	Passed
Design Logic.	Passed
Impact of the exchange rate.	Passed
Oracle Calls.	Passed
Cross-function race conditions.	Passed
Fallback function security.	Passed
Safe Open Zeppelin contracts and implementation usage.	Passed

Whitepaper-Website-Contract correlation.	Passed
Front Running.	Passed

Testing Summary

PASS

BLOCK SOLUTIONS BELIEVES

This smart contract passes the security qualifications to be listed on digital assets exchanges.

27th January, 2025

YOUR SCORE
100%

Quick Stats:

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	Assert () misuse	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	“Out of Gas” Attack	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: Passed

Executive Summary

According to the standard audit assessment, Customer`s solidity smart contract is Well-secured. Again, it is recommended to perform an Extensive audit assessment to bring a more assured conclusion.

We used various tools like Mythril, Slither and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Quick Stat section.
We found 0 critical, 1 high, 0 medium and 0 low level issues.

Code Quality

The PURRPAWS Smart Contract protocol consists of one smart contract. It has other inherited contracts like ERC20, Ownable. These are compact and well written contracts. Libraries used in PURRPAWS Smart Contract are part of its logical algorithm. They are smart contracts which contain reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in protocol. The BLOCKSOLUTIONS team has **not** provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.
Overall, the code is not commented. Commenting can provide rich documentation for functions, return variables and more.

Documentation

As mentioned above, it's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic. We were given a PURRPAWS Smart Contract smart contract code in the form of File.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well-known industry standard open-source projects. And even core code blocks are written well and systematically. This smart contract does not interact with other external smart contracts.

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
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
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Conclusion

The Smart Contract code passed the audit. We were given a contract code. And we have used all possible tests based on given objects as files. Since possible test cases can be unlimited for such extensive smart contract protocol, hence we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything. Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in Quick Stat section of the report. Audit report contains all found security vulnerabilities and other issues in the reviewed code.
Security state of the reviewed contract is "Well Secured".

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally, follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.