

# HEXNINJA AUDITS



Security Assessment  
**Pure Wallet Token**

November 17, 2022



# Table of Contents

## **1 Assessment Summary**

## **2 Technical Findings Summary**

## **3 Project Overview**

### 3.1 Token Summary

### 3.2 Risk Analysis Summary

### 3.3 Main Contract Assessed

## **4 Smart Contract Risk Checks**

### 4.1 Mint Check

### 4.2 Fees Check

### 4.3 Blacklist Check

### 4.4 MaxTx Check

### 4.5 Pause Trade Check

## **5 Contract Ownership**

## **6 Liquidity Ownership**

## **7 KYC Check**

## **8 Smart Contract Vulnerability Checks**

### 8.1 Smart Contract Vulnerability Details

### 8.2 Smart Contract Inheritance Details

### 8.3 Smart Contract Privileged Functions

## **9 Assessment Results and Notes(Important)**

## **10 Social Media Check(Informational)**

## **11 Technical Findings Details**

## **12 Disclaimer**



# Assessment Summary

This report has been prepared for Pure Wallet Token on the Binance network. CFGNINJA provides both client-centered and user-centered examination of the smart contracts and their current status when applicable. This report represents the security assessment made to find issues and vulnerabilities on the source code along with the current liquidity and token holder statistics of the protocol.

A comprehensive examination has been performed, utilizing Cross Referencing, Static Analysis, In-House Security Tools, and line-by-line Manual Review.






The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Inspecting liquidity and holders statistics to inform the current status to both users and client when applicable.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Verifying contract functions that allow trusted and/or untrusted actors to mint, lock, pause, and transfer assets.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Thorough line-by-line manual review of the entire codebase by industry experts.








# Technical Findings Summary

## Classification of Risk

Severity	Description
 Critical	Risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
 Major	Risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
 Medium	Risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform
 Minor	Risks can be any of the above but on a smaller scale. They generally do not compromise the overall integrity of the Project, but they may be less efficient than other solutions.
 Informational	Errors are often recommended to improve the code's style or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

## Findings

Severity	Found	Pending	Resolved
 Critical	0	0	0
 Major	1	1	0
 Medium	0	0	0
 Minor	1	1	0
 Informational	0	0	0
Total	2	2	0





# Project Overview

## Token Summary

Parameter	Result
Address	
Name	Pure Wallet
Token Tracker	Pure Wallet (Pure)
Decimals	18
Supply	11,000,000,000,000
Platform	Binance
compiler	v0.8.11+commit.d7f03943
Contract Name	Dogwars
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	<a href="https://bscscan.com/address/">https://bscscan.com/address/</a>
Payment Tx	0x810511b27ae4b4637125ec8f1aa1dfd06e66c071b0c5fc977bf393fb980803d8



# Project Overview

## Risk Analysis Summary

Parameter	Result
Buy Tax	0%
Sale Tax	0%
Is honeypot?	Clean
Can edit tax?	No
Is anti whale?	No
Is blacklisted?	No
Is whitelisted?	No
Holders	12
Security Score	90/100
Auditor Score	90/100
Confidence Level	Low

The following quick summary it's added to the project overview; however, there are more details about the audit and its results. Please read every detail.



## MainNet Contract was Not Assessed

### TestNet Contract Assessed Contract Name

Name	Contract	Live
Pure Wallet	0x708F4e66EB0e3ee2dD0091aa8212be7a63efa04f	no

### Solidity Code Provided

SolID	File Sha-1	FileName
staking	018e17a034e2657011d89a43a743d2694d5b36c5	staking.sol



# Mint Check

**The project owners of Pure Wallet do not have a mint function in the contract, owner cannot mint tokens after initial deploy.**

**The Project has a Total Supply of 11,000,000,000,000 and cannot mint any more than the Max Supply.**

Mint Notes:

Auditor Notes:

Project Owner Notes:





# Fees Check

**The project owners of Pure Wallet do not have the ability to set fees higher than 25% .**

**The team May have fees defined; however, they can't set those fees higher than 25% or may not be able to configure the same.**

**Tax Fee Notes:**

**Auditor Notes:** The contract currently has 0% buy and 0% sale taxes, and cannot be set.

**Project Owner Notes:**



**Fees can be changed up to a maximum of 25%**



# Blacklist Check

**The project owners of Pure Wallet do not have a blacklist function their contract.**

**The Project allow owners to transfer their tokens without any restrictions.**

**Token owner cannot blacklist the contract: Malicious or compromised owners can trap contracts relying on tokens with a blacklist.**

Blacklist Notes:

Auditor Notes:

Project Owner Notes: undefined



# MaxTx Check

**The Project Owners of Pure Wallet cannot set max tx amount**

**The Team allows any investors to swap, transfer or sell their total amount if needed.**

MaxTX Notes:

Auditor Notes:

Project Owner Notes:

Project Has No MaxTX



# Pause Trade Check

**The Project Owners of Pure Wallet don't have the ability to stop or pause trading.**

**The Team has done a great job to avoid stop trading, and investors has the ability to trade at any given time without any problems**

Pause Trade Notes:

Auditor Notes:

Project Owner Notes:



Owner can't pause trading



# Contract Ownership

**The contract ownership of Pure Wallet is not currently renounced. The ownership of the contract grants special powers to the protocol creators, making them the sole addresses that can call sensible ownable functions that may alter the state of the protocol.**

**The current owner is the address which can be viewed: [HERE](#)**

**The owner wallet has the power to call the functions displayed on the privileged functions chart below, if the owner's wallet is compromised, they could exploit these privileges.**

**We recommend the team renounce ownership at the right time, if possible, or gradually migrate to a timelock with governing functionalities regarding transparency and safety considerations.**

**We recommend the team use a Multisignature Wallet if the contract is not going to be renounced; this will give the team more control over the contract.**



# Liquidity Ownership

Most of the liquidity is currently locked; the lock can be seen here:

Liquidity Locker Link can be viewed from:  
[HERE](#)





# KYC Information

**The Project Owners of Pure Wallet have provided KYC Documentation.**

**KYC Certificated can be found on the Following:  
KYC Data**

KYC Information Notes:

Auditor Notes: KYC is Pending as customer dont have passports from China.

Project Owner Notes:



# Smart Contract Vulnerability Checks

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	staking.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	staking.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	staking.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	staking.sol	L: 3 C: 0
SWC-104	Medium	Unchecked Call Return Value.	staking.sol	L: 412 C: 27
SWC-105	Pass	Unprotected Ether Withdrawal.	staking.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	staking.sol	L: 0 C: 0
SWC-107	Low	Read of persistent state following external call.	staking.sol	L: 412 C: 27
SWC-108	Low	State variable visibility is not set..	staking.sol	L: 218 C: 9
SWC-109	Pass	Uninitialized Storage Pointer.	staking.sol	L: 0 C: 0
SWC-110	Pass	Assert Violation.	staking.sol	L: 0 C: 0
SWC-111	Pass	Use of Deprecated Solidity Functions.	staking.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	staking.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-113	Pass	Multiple calls are executed in the same transaction.	staking.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	staking.sol	L: 0 C: 0
SWC-115	Pass	Authorization through tx.origin.	staking.sol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	staking.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	staking.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	staking.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	staking.sol	L: 0 C: 0
SWC-120	Pass	Potential use of block.number as source of randomness.	staking.sol	L: 0 C: 0
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	staking.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	staking.sol	L: 0 C: 0
SWC-123	Low	Requirement Violation.	staking.sol	L: 421 C: 26, L: 192 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	staking.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	staking.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	staking.sol	L: 0 C: 0



ID	Severity	Name	File	location
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	staking.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	staking.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	staking.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U+202E).	staking.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	staking.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	staking.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	staking.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	staking.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	staking.sol	L: 0 C: 0
SWC-136	Pass	Unencrypted Private Data On-Chain.	staking.sol	L: 0 C: 0

We scan the contract for additional security issues using MYTHX and industry-standard security scanning tools.



# Smart Contract Vulnerability Details

## SWC-103 - Floating Pragma.

### CWE-664: Improper Control of a Resource Through its Lifetime.

#### References:

#### Description:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

#### Remediation:

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen.

Pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package. Otherwise, the developer would need to manually update the pragma in order to compile locally.

#### References:

Ethereum Smart Contract Best Practices - Lock pragmas to specific compiler version.



# Smart Contract Vulnerability Details

## SWC-104 - Unchecked Call Return Value.

### CWE-252: Unchecked Return Value.

#### Description:

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behaviour in the subsequent program logic.

#### Remediation:

If you choose to use low-level call methods, make sure to handle the possibility that the call will fail by checking the return value.

#### References:

Ethereum Smart Contract Best Practices - Handle errors in external calls.





# Smart Contract Vulnerability Details

## SWC-107 - Reentrancy.

### CWE-841: Improper Enforcement of Behavioral Workflow.

#### Description:

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

#### Remediation:

The best practices to avoid Reentrancy weaknesses are: Make sure all internal state changes are performed before the call is executed. This is known as the Checks-Effects-Interactions pattern Use a reentrancy lock.

#### References:

Ethereum Smart Contract Best Practices - Reentrancy



# Smart Contract Vulnerability Details

## SWC-108 - State Variable Default Visibility

### CWE-710: Improper Adherence to Coding Standards

#### Description:

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

#### Remediation:

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

#### References:

Ethereum Smart Contract Best Practices - Explicitly mark visibility in functions and state variables



# Smart Contract Vulnerability Details

## SWC-123 - Requirement Violation

### CWE-573: Improper Following of Specification by Caller

#### Description:

The Solidity `require()` construct is meant to validate external inputs of a function. In most cases, such external inputs are provided by callers, but they may also be returned by callees. In the former case, we refer to them as precondition violations. Violations of a requirement can indicate one of two possible issues:

- A bug exists in the contract that provided the external input.
- The condition used to express the requirement is too strong.

#### Remediation:

If the required logical condition is too strong, it should be weakened to allow all valid external inputs. Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

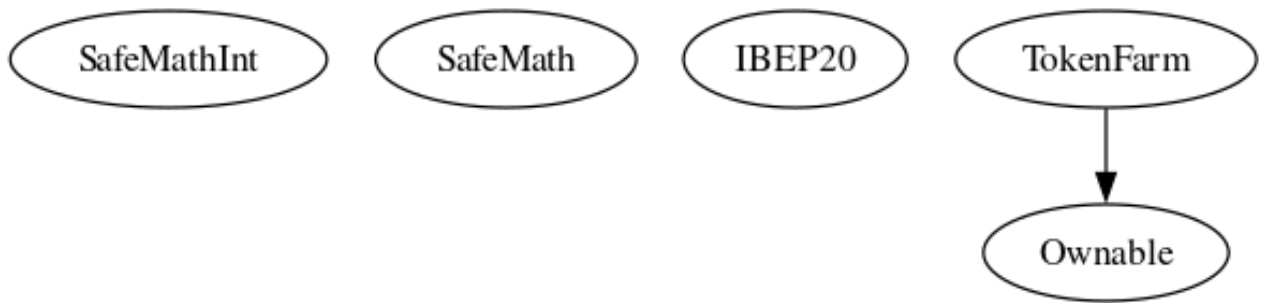
#### References:

The use of `revert()`, `assert()`, and `require()` in Solidity, and the new REVERT opcode in the EVM



# Inheritance

The contract for Pure Wallet has the following inheritance structure.



## Assessment Results

- Contract has taxes up to 15%.
- Owner can't set max tx amount.
- Owner can't pause trading.
- No high-risk Exploits/Vulnerabilities Were Found in the Source Code.
- Contract is a Staking platform
- Contract has been developed by Anoop and follow the coding best practices, we have fully tested the code and its functionalities.

## Audit Fail



# Smart Contract Advance Checks



ID	Severity	Name	Result	Status
Pure-01	Minor	Potential Sandwich Attacks.	Fail	Acknowledged
Pure-02	Informational	Function Visibility Optimization	Pass	Not Found
Pure-03	Minor	Lack of Input Validation.	Pass	Not Found
Pure-04	Major	Centralized Risk In addLiquidity.	Pass	Not Found
Pure-05	Major	Missing Event Emission.	Pass	Not Found
Pure-06	Minor	Conformance with Solidity Naming Conventions.	Pass	Not Found
Pure-07	Minor	State Variables could be Declared Constant.	Pass	Not Found
Pure-08	Major	Dead Code Elimination.	Pass	Not Found
Pure-09	Major	Third Party Dependencies.	Pass	Not Found
Pure-10	Major	Initial Token Distribution.	Fail	Pending
Pure-11	Informational	excludeFromDividends can be called and is missing an Include Function to revert action.	Pass	Not Found
Pure-12	Major	Centralization Risks In The X Role	Pass	Not Found
Pure-13	Informational	Extra Gas Cost For User..	Pass	Not Found

During our assessment review, we manually check against the following test cases, this help us identify potential missing items from any automated tools. Is an additional safety measure to ensure the contract is safe and follow industry standards.





## Pure-01 | Potential Sandwich Attacks.

Category	Severity	Location	Status
Security	 Minor	staking.sol: 470,13	 Acknowledged

### Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction being attacked) a transaction to sell the asset. The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

- swapExactTokensForETHSupportingFeeOnTransferTokens()
- addLiquidityETH()

### Remediation



We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

### References:

What Are Sandwich Attacks in DeFi – and How Can You Avoid Them?.



## Pure-10 | Initial Token Distribution.

Category	Severity	Location	Status
Centralization / Privilege	 Major	staking.sol: 559,15	 Pending

### Description

All of the Pure Wallet tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the deployer can distribute tokens without obtaining the consensus of the community.

### Remediation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

### Project Action

Initial Token Distribution goes to the msg.sender, while reviewing the current code they have 90% of the tokens locked in Pinksale.



# Social Media Checks

Social Media	URL	Result
Twitter	<a href="https://twitter.com/pure_wallet">https://twitter.com/pure_wallet</a>	Pass
Other	<a href="https://www.instagram.com/purewalletofficial/">https://www.instagram.com/purewalletofficial/</a> , <a href="https://www.facebook.com/PureWalletOfficial/">https://www.facebook.com/PureWalletOfficial/</a> , <a href="https://discord.com/invite/jKqCm9yeTn">https://discord.com/invite/jKqCm9yeTn</a>	Pass
Website	<a href="https://www.purewallet.finance//">https://www.purewallet.finance//</a>	Pass
Telegram	<a href="https://t.me/PureWalletOfficial">https://t.me/PureWalletOfficial</a>	Pass

We recommend to have 3 or more social media sources including a completed working websites.

**Social Media Information Notes:**

**Auditor Notes:** undefined

**Project Owner Notes:**



# Appendix

## Finding Categories

### Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

### Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

### Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

### Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

### Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

### Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

### Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different requirements on the input variables than a setter function.

### Coding Best Practices

ERC 20 Coding Standards are a set of rules that each developer should follow to ensure the code meets a set of criteria and is readable by all the developers.



## Disclaimer

CFGNINJA has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocacy for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided 'as is, and CFGNINJA is under no covenant to audited completeness, accuracy, or solidity of the contracts. In no event will CFGNINJA or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by CFGNINJA are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.

