CFG NINJA AUDITS

Security Assessment

Wall Finance Staking

February 2, 2023

Table of Contents

Assessment Summary

This report has been prepared for Wall Finance Staking on the Binance Smart Chain 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	0	0	0
Medium	1	1	0
Minor	0	0	0
Informational	0	0	0
Total	1	1	0

Project Overview

Token Summary

Parameter	Result
Address	0x23cfcF44FBed56332D3b4C66CD1E61e0a6E98e56
Name	Wall Finance
Token Tracker	Wall Finance ()
Decimals	
Supply	
Platform	Binance Smart Chain
compiler	v0.6.12+commit.27d51765
Contract Name	MasterWall
Optimization	Yes with 200 runs
LicenseType	MIT
Language	Solidity
Codebase	https://bscscan.com/address/0x23cfcF44FBed56332D3b4C66CD1E61e0a6E98e56#code
Payment Tx	0x342c292bb2555cf8d6b22d800b5fecb0e0dd78cc172ec9fe3f2b947b393a4f6e

Main Contract Assessed Contract Name

Name	Contract	Live
Wall Finance	0x23cfcF44FBed56332D3b4C66CD1E61e0a6E98e56	Yes

TestNet Contract Assessed Contract Name

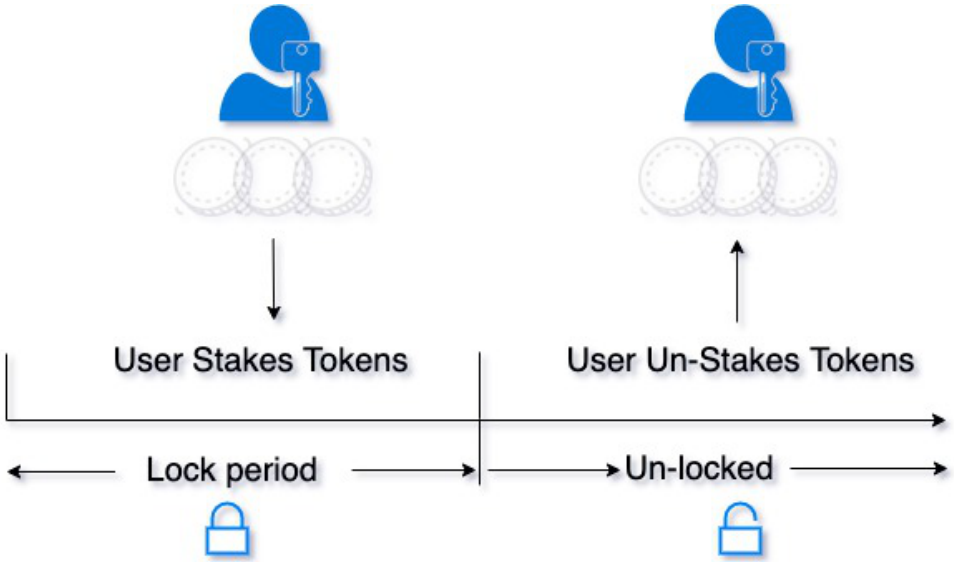
Name	Contract	Live
Wall Finance	0x67F454a9810CDB5b53cEd26f0138c2C5dB500b13	Yes

Solidity Code Provided

SolID	File Sha-1	FileName
MasterWall	d29cf8eef6d6800639dd2d24355c964538f9cf16	MasterWall.sol

What is a Staking Contract

A smart contract which allows users to stake and un-stake a specified ERC20 token. Staked tokens are locked for a specific length of time (set by the contrat owner at the outset). Once the time period has elapsed, the user can remove their tokens again.

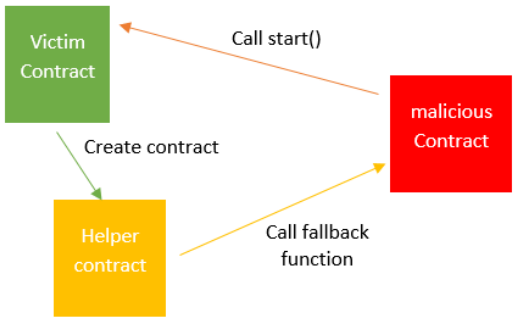


Reentrancy Check

The Project Owners of Wall Finance have implemented Reentrancy Guard Library

The Team has done a great job to avoid potential reentrancy issues in the contract.

You can read more about the reentrancy library used.
ReentrancyGuard



KYC Information

The Project Owners of Wall Finance is not KYC.

KYC Information Notes:

Auditor Notes:

Project Owner Notes:

Smart Contract Vulnerability Checks

ID	Severity	Name	File	location
SWC-100	Pass	Function Default Visibility	MasterWall.sol	L: 0 C: 0
SWC-101	Pass	Integer Overflow and Underflow.	MasterWall.sol	L: 0 C: 0
SWC-102	Pass	Outdated Compiler Version file.	MasterWall.sol	L: 0 C: 0
SWC-103	Low	A floating pragma is set.	MasterWall.sol	L: 9 C: 0, L: 226 C: 0, L: 306 C: 0, L: 398 C: 0, L: 575 C: 0, L: 602 C: 0, L: 672 C: 0
SWC-104	Pass	Unchecked Call Return Value.	MasterWall.sol	L: 0 C: 0
SWC-105	Pass	Unprotected Ether Withdrawal.	MasterWall.sol	L: 0 C: 0
SWC-106	Pass	Unprotected SELFDESTRUCT Instruction	MasterWall.sol	L: 0 C: 0
SWC-107	Low	Read of persistent state following external call.	MasterWall.sol	L: 729 C: 8
SWC-108	Pass	State variable visibility is not set..	MasterWall.sol	L: 0 C: 0
SWC-109	Pass	Uninitialized Storage Pointer.	MasterWall.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-110	Pass	Assert Violation.	MasterWall.sol	L: 0 C: 0
SWC-111	Pass	Use of Deprecated Solidity Functions.	MasterWall.sol	L: 0 C: 0
SWC-112	Pass	Delegate Call to Untrusted Callee.	MasterWall.sol	L: 0 C: 0
SWC-113	Pass	Multiple calls are executed in the same transaction.	MasterWall.sol	L: 0 C: 0
SWC-114	Pass	Transaction Order Dependence.	MasterWall.sol	L: 0 C: 0
SWC-115	Pass	Authorization through tx.origin.	MasterWall.sol	L: 0 C: 0
SWC-116	Pass	A control flow decision is made based on The block.timestamp environment variable.	MasterWall.sol	L: 0 C: 0
SWC-117	Pass	Signature Malleability.	MasterWall.sol	L: 0 C: 0
SWC-118	Pass	Incorrect Constructor Name.	MasterWall.sol	L: 0 C: 0
SWC-119	Pass	Shadowing State Variables.	MasterWall.sol	L: 0 C: 0
SWC-120	Low	Potential use of block.number as source of randonmness.	MasterWall.sol	L: 831 C: 34, L: 831 C: 62, L: 865 C: 12, L: 866 C: 69, L: 884 C: 12, L: 889 C: 35, L: 892 C: 65, L: 895 C: 31

ID	Severity	Name	File	location
SWC-121	Pass	Missing Protection against Signature Replay Attacks.	MasterWall.sol	L: 0 C: 0
SWC-122	Pass	Lack of Proper Signature Verification.	MasterWall.sol	L: 0 C: 0
SWC-123	Low	Requirement Violation.	MasterWall.sol	L: 422 C: 50, L: 747 C: 0
SWC-124	Pass	Write to Arbitrary Storage Location.	MasterWall.sol	L: 0 C: 0
SWC-125	Pass	Incorrect Inheritance Order.	MasterWall.sol	L: 0 C: 0
SWC-126	Pass	Insufficient Gas Griefing.	MasterWall.sol	L: 0 C: 0
SWC-127	Pass	Arbitrary Jump with Function Type Variable.	MasterWall.sol	L: 0 C: 0
SWC-128	Pass	DoS With Block Gas Limit.	MasterWall.sol	L: 0 C: 0
SWC-129	Pass	Typographical Error.	MasterWall.sol	L: 0 C: 0
SWC-130	Pass	Right-To-Left-Override control character (U +202E).	MasterWall.sol	L: 0 C: 0
SWC-131	Pass	Presence of unused variables.	MasterWall.sol	L: 0 C: 0
SWC-132	Pass	Unexpected Ether balance.	MasterWall.sol	L: 0 C: 0
SWC-133	Pass	Hash Collisions with Multiple Variable Length Arguments.	MasterWall.sol	L: 0 C: 0
SWC-134	Pass	Message call with hardcoded gas amount.	MasterWall.sol	L: 0 C: 0
SWC-135	Pass	Code With No Effects (Irrelevant/Dead Code).	MasterWall.sol	L: 0 C: 0

ID	Severity	Name	File	location
SWC-136	Pass	Unencrypted Private Data On-Chain.	MasterWall.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-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-120 - Weak Sources of Randomness from Chain Attributes

CWE-330: Use of Insufficiently Random Values

Description:

Solidity allows for ambiguous naming of state variables when inheritance is used. Contract A with a variable x could inherit contract B that also has a state variable x defined. This would result in two separate versions of x, one of them being accessed from contract A and the other one from contract B. In more complex contract systems this condition could go unnoticed and subsequently lead to security issues.

Shadowing state variables can also occur within a single contract when there are multiple definitions on the contract and function level.

Remediation:

Using commitment scheme, e.g. RANDAO. Using external sources of randomness via oracles, e.g. Oraclize. Note that this approach requires trusting in oracle, thus it may be reasonable to use multiple oracles. Using Bitcoin block hashes, as they are more expensive to mine.

References:

How can I securely generate a random number in my smart contract?)

When can BLOCKHASH be safely used for a random number? When would it be unsafe?

The Run smart contract.

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:

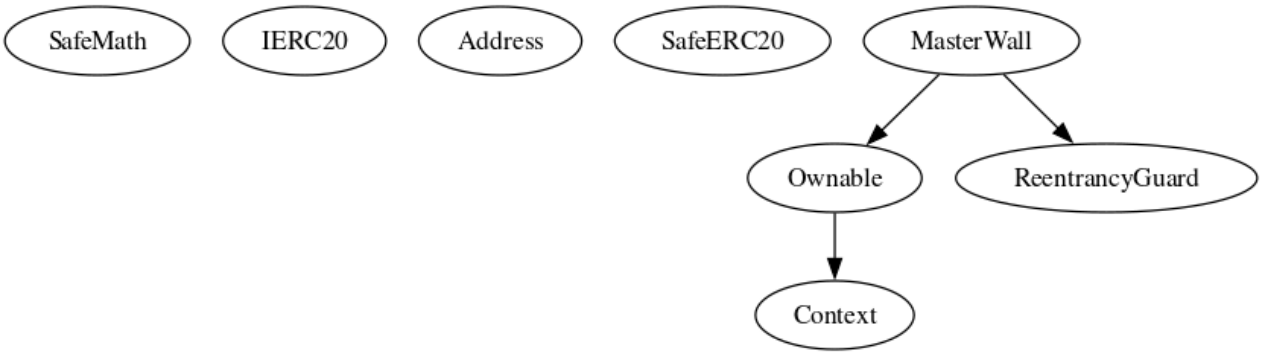
If the required logical condition is too strong, it should be weakened to allow all valid external inputs.Otherwise, the bug must be in the contract that provided the external input and one should consider fixing its code by making sure no invalid inputs are provided.

References:

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

Inheritance

The contract for Wall Finance has the following inheritance structure.



Smart Contract Advance Checks

ID	Severity	Name	Result	Status
-01	Minor	Potential Sandwich Attacks.	Pass	Not found
-02	Minor	Function Visibility Optimization	Pass	Not found
-03	Minor	Lack of Input Validation.	Pass	Not found
-04	Major	Centralized Risk In addLiquidity.	Pass	Not found
-05	Major	Missing Event Emission.	Pass	Not found
-06	Minor	Conformance with Solidity Naming Conventions.	Pass	Not Found
-07	Minor	State Variables could be Declared Constant.	Pass	Not found
-08	Major	Dead Code Elimination.	Pass	Not-Found
-09	Major	Third Party Dependencies.	Pass	Not Found
-10	Major	Initial Token Distribution.	Pass	Not found
-11	Critical	distributeTokensBetween Holders is a multisender of tokens from contract.	Pass	Not found
-12	Major	Centralization Risks In The X Role	Pass	Not Found
-13	Informational	Extra Gas Cost For User..	Pass	Not found
-14	Informational	Unnecessary Use Of SafeMath	Pass	Not found

-14 | Unnecessary Use Of SafeMath

Category	Severity	Location	Status
Logical Issue	Medium	MasterWall.sol: 20, 11	Pending

Description

The SafeMath library is used unnecessarily. With Solidity compiler versions 0.8.0 or newer, arithmetic operations
will automatically revert in case of integer overflow or underflow.
library SafeMath {
An implementation of SafeMath library is found.
using SafeMath for uint256;
SafeMath library is used for uint256 type in contract.
_balances[recipient] = _balances[recipient].add(amount);
magnifiedDividendPerShare = magnifiedDividendPerShare.add(
(amount).mul(magnitude) / totalSupply()
);
Note: Only a sample of 2 SafeMath library usage in this contract (out of 14) are shown above.

Remediation

We advise removing the usage of SafeMath library and using the built-in arithmetic operations provided by the
Solidity programming language

Project Action

Social Media Checks

Social Media	URL	Result
Twitter	https://twitter.com/wall_finance?t=Yk2s5RJvfv7cfIddm9DAFQ&s=09	Pass
Other	https://t.me/wallfinancenews,https://www.youtube.com/channel/UCiOjbxnrHKlnZ5h5MUxKmXw	Pass
Website	https://wallfinance.net/	Pass
Telegram	https://t.me/wallfinance	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:

Assessment Results

Score Results

Review	Score
Overall Score	107/100
Auditor Score	85/100
Review by Section	**Score**
Manual Scan Score	48/35
SWC Scan Score	33 /37
Advance Check Score	26 /28

The Following Score System Has been Added to this page to help understand the value of the audit, the maximun score is 100, however to attain that value the project most pass and provide all the data needed for the assessment. Our Passing Score has been changed to 80 Points, if a project does not attain 80% is an automatic failure. Read our notes and final assessment below.

Audit Passed

Assessment Results

Important Notes:

- Contract is using older compiler version, we recommened to code withe the latest compiler.
- Contract is staking contract.
- No high-risk Exploits/Vulnerabilities Were Found in the Source Code.

Auditor Score =85
Audit Passed

Appendix

Finding Categories

Centralization / Privilege

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

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimalEVM 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 howblock.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functionsbeing 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 mayresult in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to makethe 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 require statements on the input variables than a setterfunction.

Coding Best Practices

ERC 20 Conding Standards are a set of rules that each developer should follow to ensure the code meet a set of creterias 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 advocation 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.