2020

Audit Report

DETAILED AUDIT REPORT OF VANILLA CONTRACT

FARIHA ABBASI

ABOUT AUDITOR

Fariha is a Software Engineer and a Senior blockchain developer. The smart contracts and blockchain protocols have been the essence of her profession. LinkedIn GitHub

Executive Summary

This report has been prepared for Vanilla ERC20 token and the staking contract of Vanilla to discover issues and vulnerabilities in the source code of their smart contracts.

A comprehensive examination has been performed by Senior Blockchain developer and Software Engineer, Fariha Abbasi

Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the code base to ensure compliance with current best practice and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Thorough line by line manual review of the entire codebase by industry experts.

Vulnerability Classification

It categorizes issues into 3 buckets based on overall risk levels:

Critical

The code implementation does not match the specification, or it could result in the loss of funds for contract owner or users.

Medium

The code implementation does not match the specification under certain conditions, or it could affect the security standard by loosing of access control.

Low

The code implementation does not follow best practices, or use suboptimal design patterns, which may lead to security vulnerabilities further down the line.

Testing Summary

The overall score of the contract is above 95%

Type of Issues

I have scanned the source code using our proprietary static analysis tools and code-review methodologies. The following technical issues were found:

Title	Description	Issues Count	SC Line No
Integer Overflow and Underflow	An overflow/underflow happen when an arithmetic operation reaches maximum or minimum size of the type	0	
Function Incorrectness	Function Implementation does not meet the specification, leading to intentional or unintentional vulnerabilities	0	
Buffer Overflow	An attacker can write to arbitrary storage locations of a contract if array of out bound happens	0	
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished	0	
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it	0	
Timestamp Dependence	Timestamp can be influenced by minors to some degree	0	
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree	0	
“tx.origin” for authorization	“tx.origin” should not be used for authorization. Use “msg.sender” instead.	0	
Delegate call to Untrusted calling	Calling into untrusted contract is very dangerous, the target and arguments provided must be sanitized	0	
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable	0	
Function Default Visibility	Functions are public by default. A malicious user can make unauthorized or unintended state changes if a developer forgot to set the visibility	0	

Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract	0	
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement	0	
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practices	0	
Unused Variables	Unused variales reduce code quality	0	

Vulnerability Details

The Vanilla token and the staking contract have been tested across edge values and the contract is working as expected. Below are mentioned some issues and recommended notes to follow. The overall contract is not exposed to any serious vulnerabilities

Critical

No critical Issues found

Medium

All medium level issues are re-solved

Low

All low-level issues are re-solved

Manual Review Notes

Review Details

Source Contracts address

0xb97faf860045483e0c7f08c56acb31333084a988

Summary

To ensure comprehensive protection, I have analyzed the source code and manually reviewed and tested.

That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

The smart contract is not exposed to any malicious vulnerabilities. It is working as expected and no edge value issues are found.

With the final update of source code and delivery of the audit report, I have concluded that the contract sounds structurally fine and not vulnerable to any classically known anti-patterns or security issues.

The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend seeking multiple opinions, keep improving the code base, and more test coverage.

Recommendations

The contract is secure and ready to be used by users.