auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

Na'vi Moon

DATED : 9 Feb, 2024

AUDIT SUMMARY

Project name – Na'vi Moon

Date: 9 Feb, 2024

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: Passed

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	0	0	2	0
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :
The code has undergone static analysis using Slither.

Testnet version:
The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:
https://testnet.bscscan.com/address/0x4D6E4f9a6fCB53C0cb699f90e6dDCC5C326E8CAB#code

Token Information

Token Name : Na’vi Moon

Token Symbol: NAVI

Decimals: 9

Token Supply: 420000000000000

Network: Binance smart chain

Token Type: BEP-20

Token Address:
0xb66F50036B6B585cbEDe2F15243aD92fA4b370eE

Checksum:
E7265763766ad32e37ad6b85aad7331

Owner:
0x365bDE9DfCc7F5c239aF4BFBed8C01Ab1a408299
(at time of writing the audit)

Deployer:
0x9EAfbc0647F97F45cDdb62708A46e18379f9381c

AUDIT METHODOLOGY

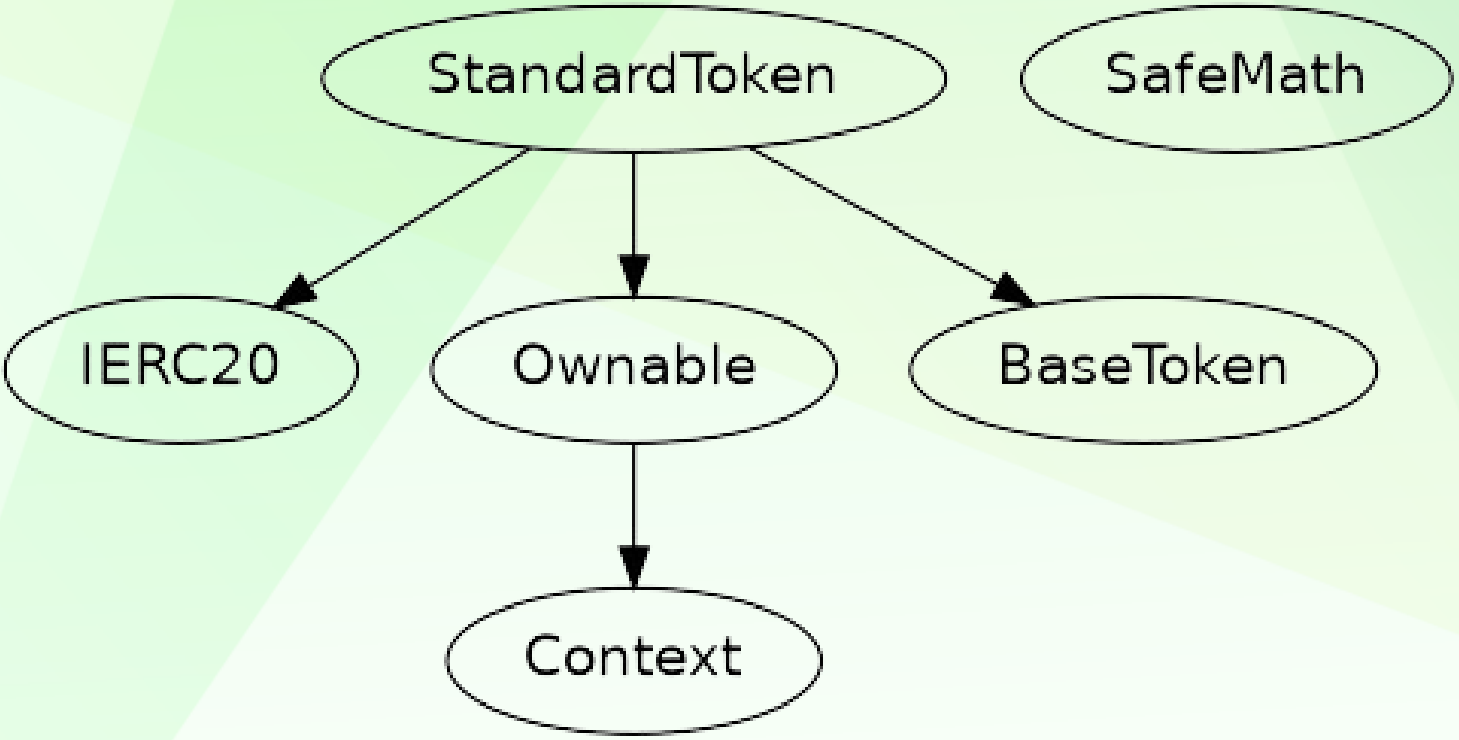
The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- [x] Return values of low-level calls
- [x] Private modifier
- [x] Multiple Sends
- [x] Using Suicide
- [x] Gas Limitand Loops
- [x] Address hardcoded
- [x] Exception Disorder
- [x] Using inline assembly
- [x] Divide before multiply
- [x] Missing Zero Address Validation
- [x] Compiler version not fixed
- [x] **Gasless Send**
- [x] Using block.timestamp
- [x] Re-entrancy
- [x] Tautology or contradiction
- [x] Timestamp Dependence
- [x] Revert/require functions
- [x] Use of tx.origin
- [x] Integer overflow/underflow
- [x] Dangerous strict equalities
- [x] Using SHA3
- [x] Using throw

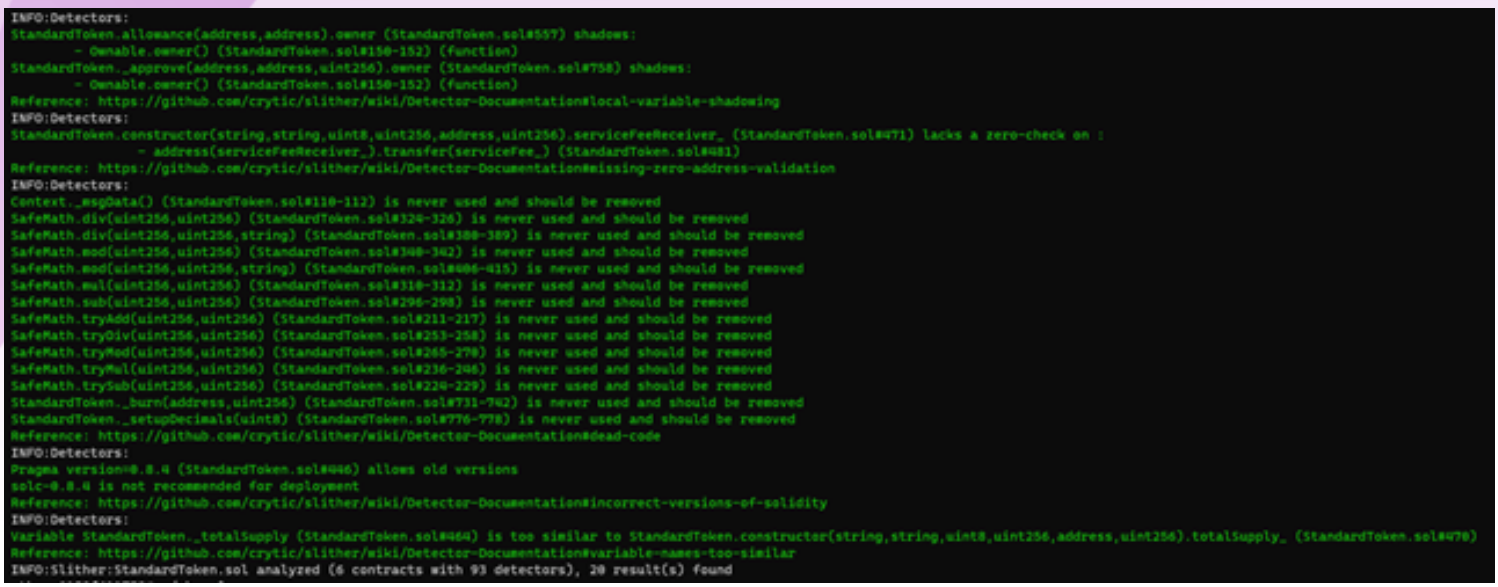
INHERITANCE TREE



STATIC ANALYSIS

A static analysis of the code was performed using Slither.

No issues were found.



```
INFO:Detectors:
StandardToken.allowance(address,address).owner (StandardToken.sol#557) shadows:
        - Ownable.owner() (StandardToken.sol#150-152) (function)
StandardToken._approve(address,address,uint256).owner (StandardToken.sol#758) shadows:
        - Ownable.owner() (StandardToken.sol#150-152) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
StandardToken.constructor(string,string,uint8,uint256,address,uint256).serviceFeeReceiver_ (StandardToken.sol#471) lacks a zero-check on :
                - address(serviceFeeReceiver_).transfer(serviceFee_) (StandardToken.sol#481)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Context._msgData() (StandardToken.sol#110-112) is never used and should be removed
SafeMath.div(uint256,uint256) (StandardToken.sol#324-326) is never used and should be removed
SafeMath.div(uint256,uint256,string) (StandardToken.sol#380-389) is never used and should be removed
SafeMath.mod(uint256,uint256) (StandardToken.sol#340-342) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (StandardToken.sol#406-415) is never used and should be removed
SafeMath.mul(uint256,uint256) (StandardToken.sol#310-312) is never used and should be removed
SafeMath.sub(uint256,uint256) (StandardToken.sol#296-298) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (StandardToken.sol#211-217) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (StandardToken.sol#253-258) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (StandardToken.sol#265-270) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (StandardToken.sol#236-246) is never used and should be removed
SafeMath.trySub(uint256,uint256) (StandardToken.sol#220-229) is never used and should be removed
StandardToken._burn(address,uint256) (StandardToken.sol#731-742) is never used and should be removed
StandardToken._setupDecimals(uint8) (StandardToken.sol#776-778) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (StandardToken.sol#446) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Variable StandardToken._totalSupply (StandardToken.sol#464) is too similar to StandardToken.constructor(string,string,uint8,uint256,address,uint256).totalSupply_ (StandardToken.sol#470)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
INFO:Slither:StandardToken.sol analyzed (6 contracts with 93 detectors), 20 result(s) found
```

FUNCTIONAL TESTING

1- Approve (passed):
https://testnet.bscscan.com/tx/0x1bc9446ead48aa567df03d7c411b90ebfd5c42f9ead386c152766f461050763d

2- Increase Allowance (passed):
https://testnet.bscscan.com/tx/0xd8f0f736201299b4dd5aa261f18e222dc30caf44e4d000d6f5842fe2f34dc944

3- Decrease Allowance (passed):
https://testnet.bscscan.com/tx/0x112affcefb8541eabf9884ab3081cd69875277a5153df9c65d797d8214f6e468

4- Transfer (passed):
https://testnet.bscscan.com/tx/0x10a8ba5c30d440e6f145c53b809a98ec38f63a5727902a6920bb14a72d42af21

POINTS TO NOTE

- The owner can renounce ownership.
- The owner can transfer ownership.
- The owner cannot mint.
- The owner cannot blacklist addresses.
- The owner cannot set high fees.

CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	0
High-Risk	0
Medium-Risk	0
Low-Risk	2
Gas Optimization / Suggestions	0

MANUAL TESTING

Centralization – Remove the safe math library

Severity: Low

Line Number: 205-416

Status: Open

Overview:

The Safe Math library is no longer needed for Solidity version 0.8 and above. This is because Solidity 0.8 includes checked arithmetic operations by default. All Safe Math's methods are now inherited into Solidity programming.

MANUAL TESTING

Centralization – Local Variable Shadowing

Severity: Low

Function: _approve and allowance

Status: Open

Overview:

```
function allowance(address owner, address spender)
        public
        view
        virtual
        override
        returns (uint256)
    {
        return _allowances[owner][spender];
    }
function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

Suggestion:
Rename the local variable that shadows another component.

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace