soken

SMART CONTRACT
SECURITY AUDIT

Multinode Finance

Scan and check this report
was posted at Soken Github

April, 2022

Website: soken.io

Table of Contents

Disclaimer

This is a comprehensive report based on our automated and manual examination of cybersecurity vulnerabilities and framework flaws. We took into consideration smart contract based algorithms, as well. Reading the full analysis report is essential to build your understanding of project's security level. It is crucial to take note, though we have done our best to perform this analysis and report, that you should not rely on the our research and cannot claim what it states or how we created it. Before making any judgments, you have to conduct your own independent research. We will discuss this in more depth in the following disclaimer - please read it fully.

DISCLAIMER: You agree to the terms of this disclaimer by reading this report or any portion thereof. Please stop reading this report and remove and delete any copies of this report that you download and/or print if you do not agree to these conditions. This report is for non-reliability information only and does not represent investment advice. No one shall be entitled to depend on the report or its contents, and Soken and its affiliates shall not be held responsible to you or anyone else, nor shall Soken provide any guarantee or representation to any person with regard to the accuracy or integrity of the report. Without any terms, warranties or other conditions other than as set forth in that exclusion and Soken excludes hereby all representations, warrants, conditions and other terms (including, without limitation, guarantees implied by the law of satisfactory quality, fitness for purposes and the use of reasonable care and skills). The report is provided as "as is" and does not contain any terms and conditions. Except as legally banned, Soken disclaims all responsibility and responsibilities and no claim against Soken is made to any amount or type of loss or damages (without limitation, direct, indirect, special, punitive, consequential or pure economic loses or losses) that may be caused by you or any other person, or any damages or damages, including without limitations (whether innocent or negligent).

Security analysis is based only on the smart contracts. No applications or operations were reviewed for security. No product code has been reviewed.

Procedure

Our analysis contains following steps:

1. Project Analysis;

2. Manual analysis of smart contracts:

- Deploying smart contracts on any of the network(Ropsten/Rinkeby) using Remix IDE
- Hashes of all transaction will be recorded
- Behaviour of functions and gas consumption is noted, as well.

3. Unit Testing:

- Smart contract functions will be unit tested on multiple parameters and under multiple conditions to ensure that all paths of functions are functioning as intended.
- In this phase intended behaviour of smart contract is verified.
- In this phase, we would also ensure that smart contract functions are not consuming unnecessary gas.
- Gas limits of functions will be verified in this stage.

4. Automated Testing:

- Mythril
- Oyente
- Manticore
- Solgraph

Terminology

We categorize the finding into 4 categories based on their vulnerability:

- Low-severity issue — less important, must be analyzed
- Medium-severity issue — important, needs to be analyzed and fixed
- High-severity issue —important, might cause vulnerabilities, must be analyzed and fixed
- Critical-severity issue —serious bug causes, must be analyzed and fixed.

Limitations

The security audit of Smart Contract cannot cover all vulnerabilities. Even if no vulnerabilities are detected in the audit, there is no guarantee that future smart contracts are safe. Smart contracts are in most cases safeguarded against specific sorts of attacks. In order to find as many flaws as possible, we carried out a comprehensive smart contract audit. Audit is a document that is not legally binding and guarantees nothing.

Token Contract Details for 08.04.2022

Contract Name: **multinode-finance-protocol**

Audit Details

Project Name: **Multinode Finance**

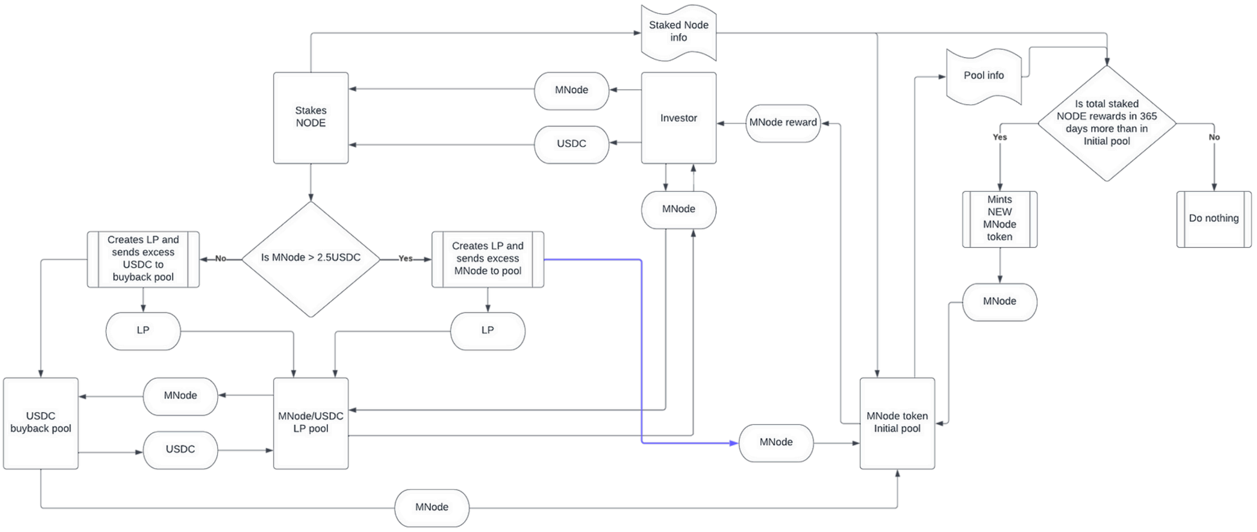
Language: **Solidity**

Compiler Version: **v0.8.12**

Blockchain: **Avalanche**

About the project

Multinode.Finance nodes on the AVAX network.
In the picture below, you can see how our protocol works. As you see, there is no loophole where funds could get out of protocol. There are no hidden tax fees or reward fees which go outwards. All sell taxes are brought back to treasury and none of them goes somewhere else



Social Profiles

Project Website: **https://www.multinode.finance/**

Project Telegram: **https://t.me/multinodefinance**

Project Twitter: **https://twitter.com/mnodefinance**

Project Discord: **https://discord.gg/6Rat52GBXm**

Project Youtube: **https://www.youtube.com/channel/UCd9fQQTHdHRaRJrJ8vqefRA**

Project Medium: **https://financenode.medium.com**

Audit Scope

multinode-finance-protocol/packages/smart-contracts/contracts/

- acl
  - ACL.sol
  - ACLControlled.sol
  - IACL.sol
- mocks
  - ERC20Mock.sol
  - Imports.sol
  - TestProtocolV2.sol
- protocol
  - IMultinodeProtocol.sol
  - MultinodeProtocol.sol
  - MultinodeProtocolStorage.sol

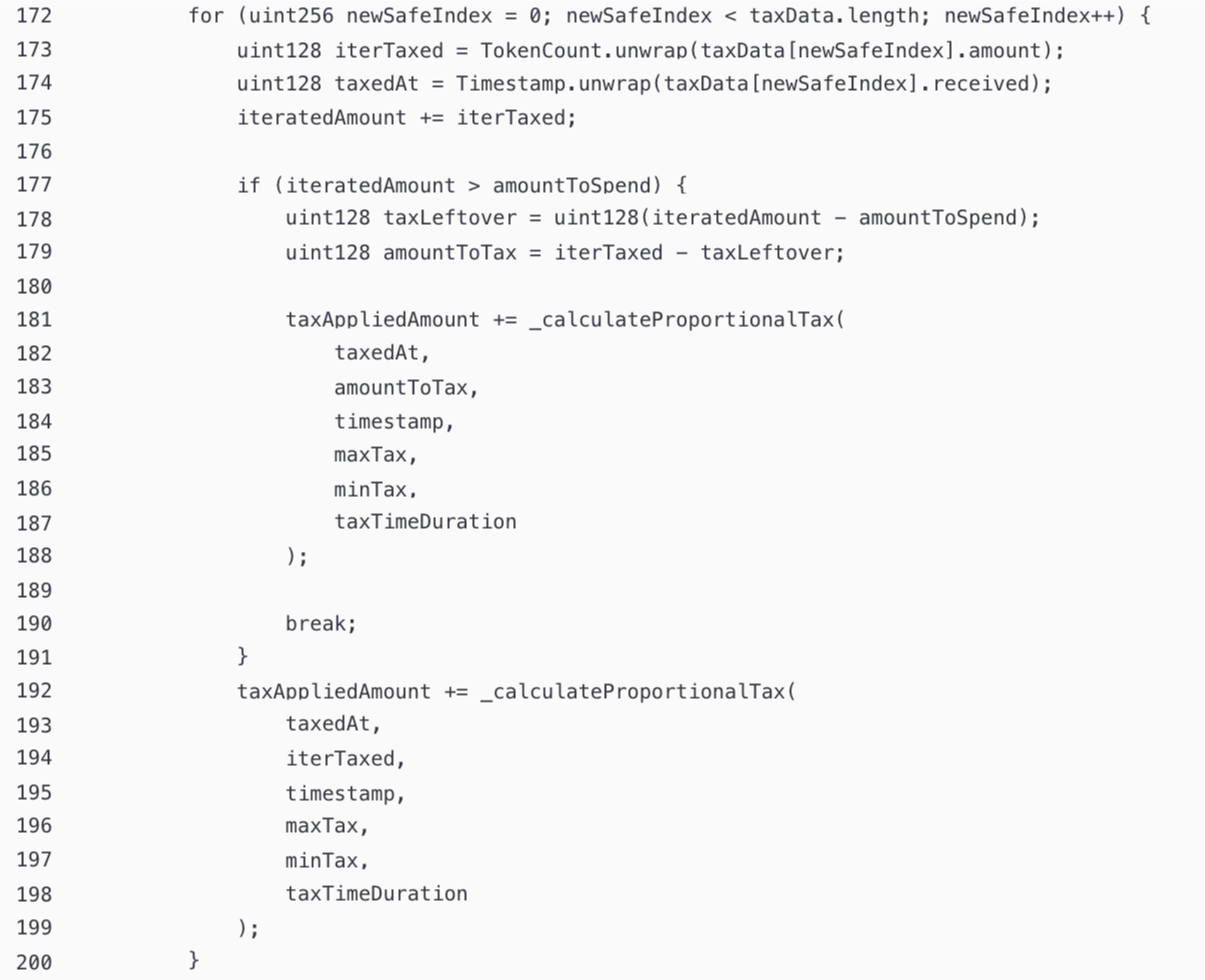
- sale
  - ITokenSale.sol
  - TokenSale.sol
- token
  - IMultinodeToken.sol
  - MultinodeToken.sol

Vulnerabilities checking

Issue Description	Checking Status
Compiler Errors	Completed
Delays in Data Delivery	Completed
Re-entrancy	Completed
Transaction-Ordering Dependence	Completed
Timestamp Dependence	Completed
Shadowing State Variables	Completed
DoS with Failed Call	Completed
DoS with Block Gas Limit	Completed
Outdated Complier Version	Completed
Assert Violation	Completed
Use of Deprecated Solidity Functions	Completed
Integer Overflow and Underflow	Completed
Function Default Visibility	Completed
Malicious Event Log	Completed
Math Accuracy	Completed
Design Logic	Completed
Fallback Function Security	Completed
Cross-function Race Conditions	Completed
Safe Zeppelin Module	Completed

Security Issues

1) LOOP CONSUMING EXCESSIVE GAS : MultinodeToken.sol



```
        for (uint256 newSafeIndex = 0; newSafeIndex < taxData.length; newSafeIndex++) {
            uint128 iterTaxed = TokenCount.unwrap(taxData[newSafeIndex].amount);
            uint128 taxedAt = Timestamp.unwrap(taxData[newSafeIndex].received);
            iteratedAmount += iterTaxed;

            if (iteratedAmount > amountToSpend) {
                uint128 taxLeftover = uint128(iteratedAmount - amountToSpend);
                uint128 amountToTax = iterTaxed - taxLeftover;

                taxAppliedAmount += _calculateProportionalTax(
                    taxedAt,
                    amountToTax,
                    timestamp,
                    maxTax,
                    minTax,
                    taxTimeDuration
                );

                break;
            }
            taxAppliedAmount += _calculateProportionalTax(
                taxedAt,
                iterTaxed,
                timestamp,
                maxTax,
                minTax,
                taxTimeDuration
            );
        }
```

Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of Gas consumed in a block. If array.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed. for (uint256 i = 0; i < array.length ; i++) {cosltyFunc();} This becomes a security issue, if an external actor influences array.length.

E.g., if an array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.

1) USE OF FLOATING PRAGMA : ACL.sol, ACLControlled.sol, IACL.sol, ERC20Mock.sol, Imports.sol, TestProtocolV2.sol, IMultinodeProtocol.sol, MultinodeProtocol.sol. MultinodeProtocolStorage.sol, ITokenSale.sol, TokenSale.sol, IMultinodeToken.sol, MultinodeToken.sol

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version. The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described, i.e., ***pragma solidity ^0.4.17; not recommended -> compiles with 0.4.17 and above pragma***
solidity 0.8.12; recommended -> compiles with 0.8.12 only

Conclusion

Smart contracts are free from any medium or high-severity issues.

NOTE: Please check the disclaimer above and note, that audit makes no statements or warranties on business model, investment attractiveness or code sustainability.

Soken Contact Info

Website: www.soken.io

Mob: (+1)416-875-4174

32 Britain Street, Toronto, Ontario, Canada

Telegram: @team_soken

GitHub: sokenteam

Twitter: @soken_team