#HashEx
BLOCKCHAIN SECURITY

Node.sys Presale

smart contracts final audit report

April 2024

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Node.sys team to perform an audit of their smart contracts. The audit was conducted between 01/04/2024 and 03/04/2024.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available in @sokol/nodesys.presale Gitlab repository and was audited after the commit d9fd0a8c.

Update. The Node.sys team has responded to this report. The updated contracts are available in the Binance Smart Chain testnet at 0x9034a12Ed1dCe4C59BD6b700ED498383cbf65B8B.

2.1 Summary

Project name	Node.sys Presale
URL	https://nodesys.io
Platform	Binance Smart Chain
Language	Solidity
Centralization level	High
Centralization risk	Medium

2.2 Contracts

Name	Address
Consensus	
PresaleNodesys	

3. Project centralization risks

The presale parameters and collected payments are managed by a consensus decision with a threshold of confirmations from consensus' owners.

Cc7CR16 Removing owners from decision making process

Any `seeMinCofReq()` number of owners can exclude other owners from consensus mechanism.

Cc8CR17 Authorized functions

The collected funds can be transferred out by a consensus transaction.

The contract manager can add 99% discount code to reduce the sale price.

4. Found issues

Medium	1 (10%)
Low	4 (40%)
Info	5 (50%)

Cc7. Consensus

ID	Severity	Title	Status
Cc7I18	Low	Default visibility of state variables	Acknowledged
Cc7I16	Info	Lack of documentation (NatSpec)	Acknowledged
Cc7I17	Info	Typographical error	Resolved

Cc8. PresaleNodesys

ID	Severity	Title	Status
Cc8I1e	Medium	Discount codes are publicly available	Acknowledged
Cc8I1d	Low	Gas optimizations	Partially fixed
Cc8I1c	Low	Default visibility of state variables	Resolved
Cc8I1f	Low	Result of token transfer is not checked	Partially fixed

Cc8I19	Info	Max purchase per transaction constraint can be circumvented	Acknowledged
Cc8I1a	Info	Lack of documentation (NatSpec)	Acknowledged
Cc8I1b	Info	Typographical error	Acknowledged

5. Contracts

Cc7. Consensus

Overview

Governance contract to authorize access to certain functions with threshold confirmations.

Issues

Cc7I18 Default visibility of state variables

Low Acknowledged

Several state variables have been declared without explicit visibility. In Solidity, the default visibility for state variables is `internal`. However, relying on default visibility can lead to misunderstandings and potential security vulnerabilities if not carefully considered and documented.

- `confirmationsRequired` (uint256)
- `GRASE_PERIOD` (uint256)
- `queue` (mapping from bytes32 to bool)
- `isOwner` (mapping from address to bool)

Cc7I16 Lack of documentation (NatSpec)

Info Acknowledged

We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Cc7I17 Typographical error

Info Resolved

The term "GRASE_PERIOD" is used in the contract, which is presumably a typographical error. The correct term should be "GRACE_PERIOD". Misnaming variables can lead to confusion for

developers, maintainers, and auditors, potentially obscuring the intent and functionality of the code.

Cc8. PresaleNodesys

Overview

A simple sale of NYS ERC-20 tokens for USDT ERC-20 payment tokens with an adjustable price. Additionally, a discount up to 99% can be applied to individual purchases by using special string codes. The sale can be stopped by a consensus decision of the owners at any moment by withdrawing sale tokens, half of which will be burnt.

Issues

Cc8I1e Discount codes are publicly available

Medium Acknowledged

A discount up to 99% can be applied to the purchases by using one of the stored codes. The codes and their discount amounts are governed by the `manager()` account. Those stored codes are reusable and can be obtained from the transaction history. If the codes were meant to be a single-use-only, they could still be front-run.

Cc8I1d Gas optimizations

Low Partially fixed

1. Multiple reads from storage in the `withdrawNYS()` function: balance of the NYS token.

2. Visibility modifier for functions `buyNYS()`, `changeManager()`, `changeMaximalAmount()` `changeMinimalAmount()`, `priceChange()` could be changed to `external` from `public`.

3. There is no need to explicitly store a timestamp in the event, as timestamp information is inherently included in the event metadata by default.

Cc8l1c Default visibility of state variables

Low Resolved

Several state variables have been declared without explicit visibility. In Solidity, the default visibility for state variables is `internal`. However, relying on default visibility can lead to misunderstandings and potential security vulnerabilities if not carefully considered and documented.

- `_minimalAmount` (uint256)
- `_maximalAmount` (mapping from address to bool)
- `_priceNYS` (uint256)
- `_manager` (address)

Cc8l1f Result of token transfer is not checked

Low Partially fixed

The contract does not check the returned results of the ERC20 transfer function.

```
    function withdrawNYS(address _owner) external onlyConsensus{
        require(isOwnerAdrress(_owner), "Not an owner");
        uint256 amount = NYS.balanceOf(address(this)) / 2;
        NYS.transfer(_owner, amount);

        uint256 amountBurn = NYS.balanceOf(address(this));
        NYS.transfer(address(1), amountBurn);

        emit WithdrawNYS(_owner, amount, amountBurn, block.timestamp);
    }
```

The ERC20 token standard mandates that the token transfer function should return a boolean value indicating the success or failure of the token transfer. Typically, tokens are designed to always return true, with the transaction failing if the transfer is unsuccessful. However, it is considered best practice to robustly handle the return values to ensure reliability.

Additionally, it is important to note that some implementations of the ERC20 token do not adhere strictly to the ERC20 standard and might not return a boolean upon transfer.

Consequently, to accommodate all scenarios, it is advisable to utilize a library that addresses these variations, such as OpenZeppelin's SafeERC20, which provides a more secure and standardized approach to handling ERC20 token transfers.

Recommendation

Use OpenZeppelin's SafeERC20 library to handle token transfers.

Cc8I19 Max purchase per transaction constraint can be circumvented

● Info ⊘ Acknowledged

The maximum purchase constraint sets the maximum purchase value only for a single transaction. The same address can make several purchases, exceeding the limit. Alternatively, a contract may make several purchases in one transaction.

Cc8I1a Lack of documentation (NatSpec)

● Info ⊘ Acknowledged

We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Cc8I1b Typographical error

● Info ⊘ Acknowledged

The term "isOwnerAdrress" is used in the contract, which is presumably a typographical error. The correct term should be "isOwnerAddress". Misnaming variables can lead to confusion for developers, maintainers, and auditors, potentially obscuring the intent and functionality of the code.

6. Conclusion

1 medium, 4 low severity issues were found during the audit. 1 low issue was resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. Issue status description

Resolved. The issue has been completely fixed.

Partially fixed. Parts of the issue have been fixed but the issue is not completely resolved.

Acknowledged. The team has been notified of the issue, no action has been taken.

Open. The issue remains unresolved.

Appendix C. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

Appendix D. Centralization risks classification

Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY