#HashEx
BLOCKCHAIN SECURITY

Node.sys

smart contracts final audit report

August 2023

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 contract. The audit was conducted between 18/08/2023 and 23/08/2023.

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 at 0xee3298518447891dee5ce74c895e44b395ba706d address in the BNB Smart Chain.

2.1 Summary

Project name	Node.sys
URL	https://nodesys.io
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
Nodesys	0xEe3298518447891dEe5ce74c895e44b395ba706d
Consensus	0xEe3298518447891dEe5ce74c895e44b395ba706d

3. Found issues

Critical	1 (10%)
High	1 (10%)
Medium	1 (10%)
Low	3 (30%)
Info	4 (40%)

Ce1. Nodesys

ID	Severity	Title	Status
Ce1Ife	Low	Mint is open	Open

Ce2. Consensus

ID	Severity	Title	Status
Ce2I0a	Critical	Everyone can delete an owner	Open
Ce2I03	High	No proposal deadline	Open
Ce2I02	Medium	No safety guard for minimum confirmations	Open
Ce2I01	Low	Gas optimizations	Open
Ce2I04	Low	Function signature collision	Open
Ce2I09	Info	Lack of events	Open

Ce2I08	Info	Lack of require message	Open
Ce2Iff	Info	Default visibility	Open
Ce2I00	Info	Typos	Open

4. Contracts

Ce1. Nodesys

Overview

An ERC-20 standard token with minting function governed by list of owners, see Consensus contract.

Issues

Ce1Ife Mint is open

Low Open

At the time of writing this report, the list of owners contains 4 EOA addresses with 3 confirmations required for minting or updating owners list.

Ce2. Consensus

Overview

Governance contract to authorize access to certain functions with threshold confirmations (more than 3 required at the time of writing).

Issues

Ce2I0a Everyone can delete an owner

Critical Open

There is no access control for the `delOwner()` function. Anyone can delete an owner.

```
  function delOwner(uint indexOwner) public {
        require(indexOwner <= owners.length,"Node index cannot be higher than their
number"); // index must be less than or equal to array length
```

```
        require(owners.length -1  >= confirmationsRequired,"error minimal count owner");

        for (uint i = indexOwner; i < owners.length-1; i++){
            owners[i] = owners[i+1];
        }
        isOwner[owners[indexOwner]] = false;
        delete owners[owners.length-1];
        owners.pop();
    }
```

The contract has a check that there should be at least `confirmationsRequired` owners and a malicious actor can't delete all owners, but if there is no access for one of the rest owners, the onlyConsensus functions would be impossible to call.

Recommendation

Add the `onlyConsensus()` modifier to the function.

Ce2I03 No proposal deadline

High Open

Proposals have timestamp field, which is not used during the execution. A malicious proposal may be pended silently for an arbitrary period of time.

Recommendation

Consensus members list may be updated to contracts only, which have safety check implemented.

Ce2I02 No safety guard for minimum confirmations

Medium Open

Minimum value of `confirmationsRequired` should be added as safety guard in the `assignRequiredConf()` function. Otherwise, a malicious executed proposal can grant full access to a single address.

Recommendation

Consensus members list may be updated to contracts only, which have this safety check implemented.

Ce2I01 Gas optimizations

Low Open

1. Multiple reads from storage in the `confirm()` and `cancelConfirmation()` functions: `eps[_txId].confirmations` variable.

2. Multiple reads from storage in the `discardExecProposal()` function: `owners.length` variable.

3. Multiple reads from storage in the `delOwner()` function: `owners.length` variable.

4. Ineffective removal of array item in the `delOwner()` function, only a single storage write should be performed.

5. Code with no effect in the `delOwner()` functions: `delete owners[owners.length-1]` before `owners.pop()` should be removed.

Ce2I04 Function signature collision

Low Open

The incoming proposal is stored in form of `ExecProposal.func::string`, `ExecProposal.data::bytes`. A malicious proposal may be constructed to exploit `bytes4(keccak256(bytes(execProposal.func)))` hash collision.

Recommendation

Consensus members must calculate function signature locally before approving proposal.

Ce2I09 Lack of events

Info Open

The function `assignRequiredConf()` changes contract state but does not emit event. We recommend adding an event to simplify off-chain tracking of important value changes.

Ce2I08 Lack of require message

Info Open

The contstructor has a require statement without a message.

```
constructor(address[] memory _owners){
    require(_owners.length >= confirmationsRequired);
    ...
}
```

We recommned adding require messages to make debugging easier.

Ce2Iff Default visibility

Info Open

No explicit visibility is defined for the `confirmationsRequired`, `queue`, `isOwner` variables.

Ce2I00 Typos

Info Open

Typos reduce code readability. Typos in 'execut', 'allready'.

5. Conclusion

1 critical, 1 high, 1 medium, 3 low severity issues were found during the audit. No issues were resolved in the update.

The reviewed contracts are dependent on the list of privileged account. Users can check list via `seeOwners()` and `seeMinCofReq()` functions.

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. 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

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY