

SMART CONTRACT

Security Audit Report

Project: EGON Hold and Node Staking
Platform: Egon Blockchain
Language: Solidity
Date: September 5th, 2023

Table of contents

Introduction	4
Project Background	4
Audit Scope	5
Claimed Smart Contract Features	6
Audit Summary	7
Technical Quick Stats	8
Code Quality	9
Documentation	9
Use of Dependencies	9
AS-IS overview	10
Severity Definitions	12
Audit Findings	13
Conclusion	22
Our Methodology	23
Disclaimers	25
Appendix	
• Code Flow Diagram	26
• Slither Results Log	28
• Solidity static analysis	31
• Solhint Linter	36

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Introduction

EtherAuthority was contracted by the Hold and Node Staking team to perform the Security audit of the Hold and Node Staking smart contracts code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on September 5th, 2023.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- Hold and Node Staking is a contract that can be divided into multiples, each with unique functionalities:
 - **HoldElevator:** The standard stake reward for EGON Holders is currently held.
 - **NodeElevator:** Node contract for active validators to get rewarded since EgonCoin Blockchain has no block reward.
- Smart contracts offer various functions such as pause/unpause contracts, withdrawable revenue, withdraw, unstake and deposit.

Audit scope

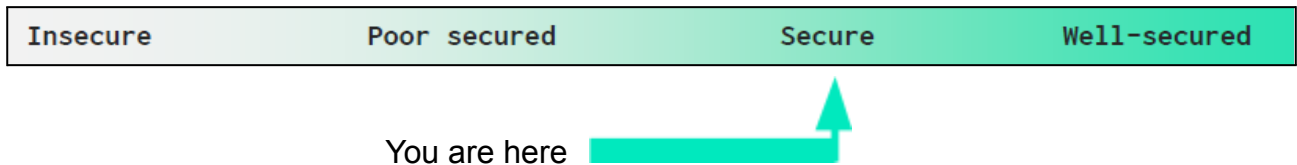
Name	Code Review and Security Analysis Report for Hold and Node Staking Smart Contracts
Platform	Egon Blockchain / Solidity
File 1	HoldElevator.sol
File 1 Online Code	0xCf4C502a2Be5E83Ae138fC88E4b69ef93795E990
Updated File 1 MD5 Hash	6841BD790E6D869D25EF54561E7A9E14
File 2	NodeElevator.sol
File 2 Online Code	0xaEbC3307309D3475309B872Fd869eD3240BdC4Aa
Updated File 2 MD5 Hash	E641D9B9050175E216B515FF62FCFF47
Audit Date	September 2nd, 2023
Revised Audit Date	September 9th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
<p>File 1 HoldElevator.sol</p> <ul style="list-style-type: none">• Premature Unstake : 15% fee from reward earned. <p><u>The Owner has control over the following functions:</u></p> <ul style="list-style-type: none">• Update packages.• Contract is paused / unpaused.• Set multiple unstake addresses.• Unstake single address.	<p>YES, This is valid.</p>
<p>File 2 NodeElevator.sol</p> <ul style="list-style-type: none">• Premature Unstake : 15% fee from reward earned. <p><u>The Owner has control over the following functions:</u></p> <ul style="list-style-type: none">• Contract is paused / unpaused.• Set addresses in WhiteListed.• Set multiple unstake addresses.• Unstake single address.• Update packages.	<p>YES, This is valid.</p>

Audit Summary

According to the standard audit assessment, Customer's solidity smart contracts are **"Secured"**. Also, these contracts contain owner control, which does not make them fully decentralized.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 2 medium, 0 low and 10 very low level issues.

We confirm that all these severities are fixed / acknowledged in the revised smart contract code.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Moderated
	High consumption 'for/while' loop	Passed
	High consumption 'storage' storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	"Short Address" Attack	Passed
	"Double Spend" Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 2 smart contract files. Smart contracts contain Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in Hold and Node Staking are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Hold and Node Staking Protocol.

The Hold and Node Staking team has not provided unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on smart contracts.

Documentation

We were given a Hold and Node Staking smart contract code in the form of a <https://testnet.egonscan.com> web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are not well commented on. but the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contracts infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

HoldElevator.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	receive	external	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	isPaused	modifier	Passed	No Issue
5	isNotPaused	modifier	Passed	No Issue
6	updatePackages	external	Passed	No Issue
7	pause	external	access only Owner	No Issue
8	unpause	external	access only Owner	No Issue
9	unstakeBulk	external	Passed	No Issue
10	unstakeSingle	external	Passed	No Issue
11	deposit	external	Passed	No Issue
12	unstake	write	Passed	No Issue
13	settleSingle	internal	Passed	No Issue
14	withdraw	external	Passed	No Issue
15	revenueOf	external	Passed	No Issue
16	withdrawableRevenueOf	write	Gas Efficiency	Refer Audit Findings
17	getUserInfo	external	Passed	No Issue
18	globalInfo	external	Passed	No Issue

NodeElevator.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	receive	external	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	isPaused	modifier	Passed	No Issue
5	isNotPaused	modifier	Passed	No Issue
6	setWhiteListed	write	access only Owner	No Issue
7	pause	external	access only Owner	No Issue
8	unpause	external	access only Owner	No Issue
9	unstakeBulk	external	Passed	No Issue
10	unstakeSingle	external	Passed	No Issue
11	updatePackages	external	Passed	No Issue
12	deposit	external	Duplicate validation in the same function	Refer Audit Findings
13	withdraw	external	Passed	No Issue
14	unstake	write	Passed	No Issue

15	settleSingle	internal	Passed	No Issue
16	revenueOf	external	Passed	No Issue
17	withdrawableRevenueOf	write	Gas Efficiency	Refer Audit Findings
18	getUserInfo	external	Passed	No Issue
19	globalInfo	external	Passed	No Issue

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No critical severity vulnerabilities were found in the contract code.

High Severity

No high severity vulnerabilities were found in the contract code.

Medium

(1) Daily percentage vulnerabilities:

HoldElevator.sol

```
function updatePackages( 48887 gas
    uint40 _invest_days,
    uint256 _dailypercentage,
    uint256 packageId
) external onlyOwner{
    packageList[packageId].dailypercentage = _dailypercentage;
    packageList[packageId].invest_days = _invest_days;
}
```

NodeElevator.sol

```
function updatePackages( 48887 gas
    uint40 _invest_days,
    uint256 _dailypercentage,
    uint256 packageId
) external onlyOwner{
    packageList[packageId].dailypercentage = _dailypercentage;
    packageList[packageId].invest_days = _invest_days;
}
```

No validation is done inside the updatePackages() function.


The `_dailypercentage` has no validation while updating by the owner, so the owner can set the daily percentage to 0 and then the user will not get a daily revenue of his deposit.

Resolution: We suggest using validation for the daily percentage variable and it must be greater than zero.

Status: **Fixed**

(2) Large user wallet array size:

HoldElevator.sol


```
function deposit(uint16 _package) external payable isNotPaused {  infinite gas
    uint256 amount = msg.value;
    require(amount >= minStake && amount <= maxStake, "Invalid amount");
    require(
        packageList[_package].invest_days > 0,
        "Out of investment days range"
    );

    User storage user = users[msg.sender];

    if (users[msg.sender].invested == 0) {
        totalUsers++;
    }
    if (user.deposits.length == 0) {
        stakingList.push(msg.sender);
    }
    user.deposits.push(
        Deposit({

```

NodeElevator.sol

```
function deposit(uint16 _package) external payable isNotPaused {  infinite gas
    uint256 amount = msg.value;
    require(amount >= minStake && amount <= maxStake, "Invalid amount");
    require(users[msg.sender].invested == 0, "Already staked");
    require(
        packageList[_package].invest_days > 0,
        "Out of investment days range"
    );
    require(isWhitelisted[msg.sender], "User is not whitelisted");
    User storage user = users[msg.sender];

    if (users[msg.sender].invested == 0) {
        totalUsers++;
    }

    if (user.deposits.length == 0) {
        stakingList.push(msg.sender);
    }

```

While depositing users added to an array is relatively straightforward, there is no code to remove that user from the array when that user has unstaked all his deposits. It will cause issues when the array size increases. It may stop the deposit functionality in the future.

Resolution: Implement a dynamic array with the "swap and pop" technique for constant-time removal of users from the array to address scalability issues.

Status: **Fixed**

Low

No low severity vulnerabilities were found in the contract code.

Very Low / Informational / Best practices:

(1) SafeMath Library: [HoldElevator.sol](#), [NodeElevator.sol](#)

SafeMath Library is used in this contract code, but if the compiler version is greater than or equal to 0.8.0, then it will not be required to be used; Solidity automatically handles overflow and underflow.

Resolution: Remove the SafeMath library and use normal math operators. It will improve code size and reduce gas consumption.

Status: **Fixed**

(2) Make variables constant: [HoldElevator.sol](#)

```
uint256 public unstakeFees = 15;
uint256 public minStake = 271*1e18;
uint256 public maxStake = 127100*1e18;
bool public paused;
```

These variables will be unchanged. So, please make it constant. It will save some gas.

Resolution: Declare those variables as constants. Just use a constant keyword.

Status: **Fixed**

(3) Missing event logs: [HoldElevator.sol](#), [NodeElevator.sol](#)

The unstakeBulk and unstakeSingle functions are executed by the owner, but they should log events.

Resolution: We suggest adding a log for these owner functions.

Status: **Fixed**

(4) Unused event:

[HoldElevator.sol](#)

```
event Register(  
    address indexed user,  
    address indexed referrer,  
    uint256 time  
);  
event Claimed(address indexed user, uint256 amount);
```

[NodeElevator.sol](#)

```
event Register(  
    address indexed user,  
    address indexed referrer,  
    uint256 time  
);  
event Claimed(address indexed user, uint256 amount);
```

Register and claimed events are defined but not used in code.

Resolution: We suggest removing unused events.

Status: **Fixed**

(5) Require message irrelevant: [HoldElevator.sol](#)

```
modifier isPaused {  
    require(paused, "Invalid User");  
    _;  
}
```

The isPaused() modifier requires that the message is irrelevant.

Resolution: We suggest setting the proper message to required.

Status: **Fixed**

(6) Variable naming is not proper:

HoldElevator.sol

```
constructor(address payable marketingAddr) {  infinite gas 2595600 gas
    ownerAddress = marketingAddr;
    packageList[0] = Package(100, 10);
    packageList[1] = Package(200, 15);
    packageList[2] = Package(350, 20);
    packageList[3] = Package(720, 25);
}

receive() external payable {  undefined gas
    //contract is able to receive funds
}

modifier onlyOwner {
    require(msg.sender == ownerAddress, "Invalid User");
    _;
}
```

NodeElevator.sol

```
constructor(address payable marketingAddr) {  infinite gas 1531800 gas
    ownerAddress = marketingAddr;
    packageList[0] = Package(360, 50);
}

receive() external payable {  undefined gas
    //contract is able to receive funds
}

modifier onlyOwner {
    require(msg.sender == ownerAddress, "Invalid User");
    _;
}
```

The constructor asks for the parameter as a marketing address, and that marketing wallet is used in the contract as the owner's wallet.

Resolution: We suggest using proper variable and variable naming.

Status: Fixed

(7) Utilizing Custom Modifiers for Enhanced Security: [HoldElevator.sol](#)

```
modifier onlyOwner {
    require(msg.sender == ownerAddress, "Invalid User");
    _;
}

function updatePackages(
    uint40 _invest_days,
    uint256 _dailypercentage,
    uint256 packageId
) external {
    require(msg.sender == ownerAddress, "Invalid user");
    packageList[packageId].dailypercentage = _dailypercentage;
    packageList[packageId].invest_days = _invest_days;
}
```

Streamline security in your smart contracts with custom modifiers. Simplify access control logic, enhance code readability, and reduce redundancy.

Resolution: Implement custom modifiers like 'onlyOwner' to centralize ownership checks. This promotes code reuse, improves code comprehension, and ensures consistent security measures.

Status: **Fixed**

(8) Gas Efficiency:

[HoldElevator.sol](#)

```
function withdrawableRevenueOf(address _addr) infinite gas
    public
    returns (uint256 value)
{
    User storage user = users[_addr];

    for (uint256 i = 0; i < user.deposits.length; i++) {
        Deposit storage dep = user.deposits[i];
        if (!dep.isUnstaked) {...
    }
}
user.last_withdraw = uint40(block.timestamp);
return value;
}
```

NodeElevator.sol

```
function withdrawableRevenueOf(address _addr) ⓘ infinite gas
    public
    returns (uint256 value)
{
    User storage user = users[_addr];

    for (uint256 i = 0; i < user.deposits.length; i++) {
        Deposit storage dep = user.deposits[i];
        if (!dep.isUnstaked) { ...
        }
    }
    user.last_withdraw = uint40(block.timestamp);
    return value;
}
```

The current behavior of the "withdrawableRevenueOf" function can result in users unnecessarily spending gas fees to execute transactions, when the user is already isUnstaked.

Resolution: Modify the "withdrawableRevenueOf" function to include a check that ensures at least one user is already isUnstaked or not.

Status: **Acknowledged**

(9) Duplicate validation in the same function: [NodeElevator.sol](#)

```
function deposit(uint16 _package) external payable isNotPaused { ⓘ infinite gas
    uint256 amount = msg.value;
    require(amount >= minStake && amount <= maxStake, "Invalid amount");
    require(users[msg.sender].invested == 0, "Already staked");
    require(
        packageList[_package].invest_days > 0,
        "Out of investment days range"
    );
    require(isWhiteListed[msg.sender], "User is not whitelisted");
    User storage user = users[msg.sender];

    if (users[msg.sender].invested == 0) {
        totalUsers++;
    }

    if (user.deposits.length == 0) {
        stakingList.push(msg.sender);
    }
}
```

If the deposit function is already checked before it is invested 0, then the next validation is not working, and function execution will fail.

Resolution: We suggest removing duplications of validation from the code.

Status: **Acknowledged**

(10) Unused struct variables:

NodeElevator.sol

```
struct User {
    bool isClaimed;
    uint256 earned;
    Deposit[] deposits;
    uint256 invested;
    uint40 last_withdraw;
    uint256 withdrawn;
}
```

HoldElevator.sol

```
struct User {
    bool isClaimed;
    uint256 earned;
    Deposit[] deposits;
    uint256 invested;
    uint40 last_withdraw;
    uint256 withdrawn;
    uint256 availableToWithdraw;
}
```

In the user structure, variables "isClaimed" and "earned" are defined but not used anywhere.

Resolution: We suggest removing unused variables from the structure.

Status: **Fixed**

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble.

Following are Admin functions:

HoldElevator.sol

- updatePackages: The owner can update packages.
- pause: contract status set paused true by the owner.
- unpause: contract status set paused false by the owner.
- unstakeBulk: The owner can set multiple unstake addresses.
- unstakeSingle: The owner can unstake a single address.

NodeElevator.sol

- setWhiteListed: Added user address in the white list by the owner.
- pause: contract status set paused true by the owner.
- unpause: contract status set paused false by the owner.
- unstakeBulk: The owner can set multiple unstake addresses.
- unstakeSingle: The owner can unstake a single address.
- updatePackages: The owner can update packages.

To make the smart contract 100% decentralized, we suggest renouncing ownership of the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a <https://testnet.egonscan.com> web link. And we have used all possible tests based on given objects as files. We had observed 2 medium issues and 10 Informational severity issues in the smart contracts. We confirm that all these severities are fixed / acknowledged in the revised smart contract code. **So, the smart contracts are ready for the mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed contract, based on standard audit procedure scope, is **“Secured”**.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

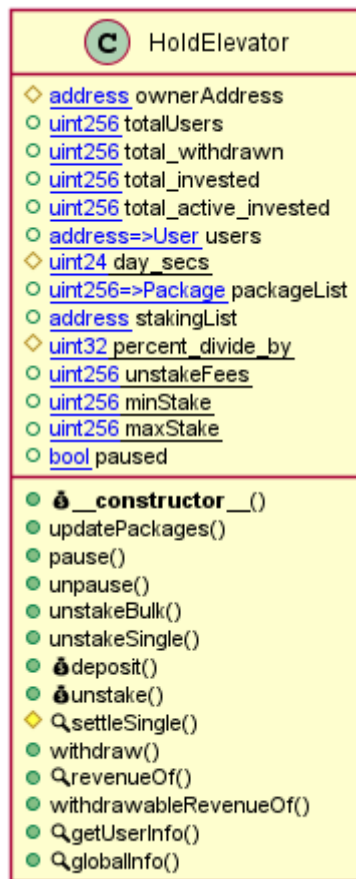
Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

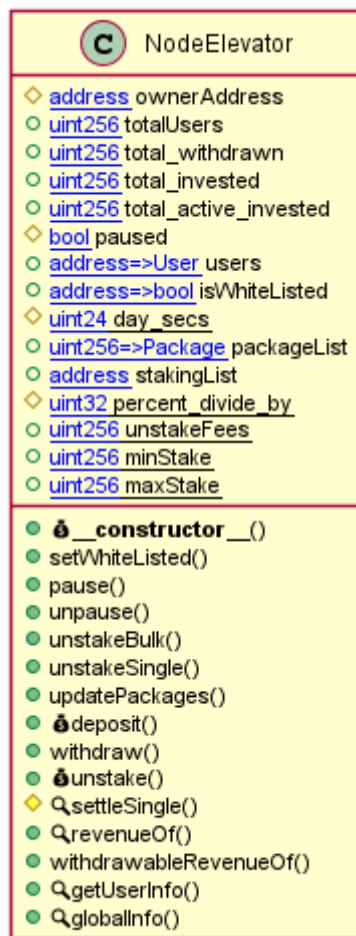
Appendix

Code Flow Diagram - Hold and Node Staking Protocol

HoldElevator Diagram



NodeElevator Diagram



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither log >> HoldElevator.sol

```
HoldElevator.constructor(address).marketingAddr (HoldElevator.sol#217) lacks a zero-check on :
- ownerAddress = marketingAddr (HoldElevator.sol#218)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

HoldElevator.unstakeBulk(uint256,uint256) (HoldElevator.sol#262-287) has external calls inside a loop: address(_user).transfer
(dep.amount) (HoldElevator.sol#275)
HoldElevator.unstakeSingle(address) (HoldElevator.sol#289-308) has external calls inside a loop: address(_user).transfer(dep.a
mount) (HoldElevator.sol#298)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

HoldElevator.unstake(uint256) (HoldElevator.sol#341-370) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(! dep.isUnstaked,Already unstaked) (HoldElevator.sol#343)
- revenue > 0 (HoldElevator.sol#347)
- block.timestamp < (dep.time + (packageList[dep.package].invest_days * day_secs)) (HoldElevator.sol#349-350)
- users[msg.sender].invested == 0 (HoldElevator.sol#367)
HoldElevator.settleSingle(address,uint256,uint256) (HoldElevator.sol#372-400) uses timestamp for comparisons
Dangerous comparisons:
- from < to (HoldElevator.sol#390)
- _at > time_end (HoldElevator.sol#388)

HoldElevator.settleSingle(address,uint256,uint256) (HoldElevator.sol#372-400) uses timestamp for comparisons
Dangerous comparisons:
- from < to (HoldElevator.sol#390)
- _at > time_end (HoldElevator.sol#388)
HoldElevator.withdrawableRevenueOf(address) (HoldElevator.sol#446-484) uses timestamp for comparisons
Dangerous comparisons:
- from < to (HoldElevator.sol#465)
- block.timestamp < (dep.time + (packageList[dep.package].invest_days * day_secs)) (HoldElevator.sol#472-473)
- block.timestamp > time_end (HoldElevator.sol#463)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

HoldElevator.unstakeBulk(uint256,uint256) (HoldElevator.sol#262-287) has costly operations inside a loop:
- total_active_invested -= dep.amount (HoldElevator.sol#277)
HoldElevator.unstakeBulk(uint256,uint256) (HoldElevator.sol#262-287) has costly operations inside a loop:
- totalUsers -- (HoldElevator.sol#282)
HoldElevator.unstakeSingle(address) (HoldElevator.sol#289-308) has costly operations inside a loop:
- total_active_invested -= dep.amount (HoldElevator.sol#301)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

SafeMath.add(uint256,uint256) (HoldElevator.sol#19-24) is never used and should be removed
SafeMath.mod(uint256,uint256) (HoldElevator.sol#137-139) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (HoldElevator.sol#153-160) is never used and should be removed
SafeMath.sub(uint256,uint256) (HoldElevator.sol#36-38) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (HoldElevator.sol#50-59) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (HoldElevator.sol#6) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Parameter HoldElevator.updatePackages(uint40,uint256,uint256)._invest_days (HoldElevator.sol#245) is not in mixedCase
Parameter HoldElevator.updatePackages(uint40,uint256,uint256)._dailypercentage (HoldElevator.sol#246) is not in mixedCase
Parameter HoldElevator.unstakeSingle(address)._user (HoldElevator.sol#289) is not in mixedCase
Parameter HoldElevator.deposit(uint16)._package (HoldElevator.sol#310) is not in mixedCase
Parameter HoldElevator.settleSingle(address,uint256,uint256)._addr (HoldElevator.sol#372) is not in mixedCase
Parameter HoldElevator.settleSingle(address,uint256,uint256)._at (HoldElevator.sol#372) is not in mixedCase
Parameter HoldElevator.revenueOf(address,uint256)._addr (HoldElevator.sol#413) is not in mixedCase
Parameter HoldElevator.revenueOf(address,uint256)._at (HoldElevator.sol#413) is not in mixedCase
Parameter HoldElevator.withdrawableRevenueOf(address)._addr (HoldElevator.sol#446) is not in mixedCase
Parameter HoldElevator.getUserInfo(address)._addr (HoldElevator.sol#489) is not in mixedCase
Variable HoldElevator.total_withdrawn (HoldElevator.sol#169) is not in mixedCase
Variable HoldElevator.total_invested (HoldElevator.sol#170) is not in mixedCase
Variable HoldElevator.total_active_invested (HoldElevator.sol#171) is not in mixedCase
Constant HoldElevator.day_secs (HoldElevator.sol#197) is not in UPPER_CASE_WITH_UNDERSCORES
Constant HoldElevator.percent_divide_by (HoldElevator.sol#200) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

```

Reentrancy in HoldElevator.unstake(uint256) (HoldElevator.sol#341-370):
  External calls:
  - address(msg.sender).transfer(withdrawable) (HoldElevator.sol#358)
  State variables written after the call(s):
  - totalUsers -- (HoldElevator.sol#368)
  - total_active_invested -= dep.amount (HoldElevator.sol#365)
  - user.availableToWithdraw = 0 (HoldElevator.sol#360)
  - dep.isUnstaked = true (HoldElevator.sol#362)
  - dep.withdrawn += (withdrawable - dep.amount) (HoldElevator.sol#363)
  - users[msg.sender].invested -= dep.amount (HoldElevator.sol#364)
  Event emitted after the call(s):
  - NewWithdrawn(msg.sender,withdrawable) (HoldElevator.sol#359)
  - Unstaked(msg.sender,dep.amount) (HoldElevator.sol#366)
Reentrancy in HoldElevator.unstakeBulk(uint256,uint256) (HoldElevator.sol#262-287):
  External calls:
  - address(_user).transfer(dep.amount) (HoldElevator.sol#275)
  State variables written after the call(s):
  - totalUsers -- (HoldElevator.sol#282)
  - total_active_invested -= dep.amount (HoldElevator.sol#277)
  - users[_user].invested -= dep.amount (HoldElevator.sol#276)
Reentrancy in HoldElevator.unstakeSingle(address) (HoldElevator.sol#289-308):
  External calls:
  - address(_user).transfer(dep.amount) (HoldElevator.sol#298)
  State variables written after the call(s):
  - totalUsers -- (HoldElevator.sol#305)
  - total_active_invested -= dep.amount (HoldElevator.sol#301)
  - users[_user].invested -= dep.amount (HoldElevator.sol#300)
Reentrancy in HoldElevator.withdraw() (HoldElevator.sol#402-411):
  External calls:
  - address(msg.sender).transfer(amount) (HoldElevator.sol#409)
  Event emitted after the call(s):
  - NewWithdrawn(msg.sender,amount) (HoldElevator.sol#410)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

HoldElevator.slitherConstructorConstantVariables() (HoldElevator.sol#164-535) uses literals with too many digits:
  - percent_divide_by = 100000 (HoldElevator.sol#200)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

HoldElevator.maxStake (HoldElevator.sol#203) should be constant
HoldElevator.minStake (HoldElevator.sol#202) should be constant
HoldElevator.unstakeFees (HoldElevator.sol#201) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

HoldElevator.ownerAddress (HoldElevator.sol#167) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
HoldElevator.sol analyzed (2 contracts with 84 detectors), 43 result(s) found

```

Slither log >> NodeElevator.sol

```

NodeElevator.constructor(address).marketingAddr (NodeElevator.sol#217) lacks a zero-check on :
  - ownerAddress = marketingAddr (NodeElevator.sol#218)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

NodeElevator.unstakeBulk(uint256,uint256) (NodeElevator.sol#253-270) has external calls inside a loop: address(_user).transfer
(users[_user].invested) (NodeElevator.sol#263)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

NodeElevator.deposit(uint16) (NodeElevator.sol#296-329) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool,string)(users[msg.sender].invested == 0,Allready staked) (NodeElevator.sol#299)
NodeElevator.unstake(uint256) (NodeElevator.sol#344-373) uses timestamp for comparisons
  Dangerous comparisons:
  - require(bool,string)(! dep.isUnstaked,Allready unstaked) (NodeElevator.sol#346)
  - revenue > 0 (NodeElevator.sol#350)
  - block.timestamp < (dep.time + (packageList[dep.package].invest_days * day_secs)) (NodeElevator.sol#352-353)
  - users[msg.sender].invested == 0 (NodeElevator.sol#370)
NodeElevator.settleSingle(address,uint256,uint256) (NodeElevator.sol#375-401) uses timestamp for comparisons
  Dangerous comparisons:
  - from < to (NodeElevator.sol#391)
  - _at > time_end (NodeElevator.sol#389)
NodeElevator.withdrawableRevenueOf(address) (NodeElevator.sol#437-475) uses timestamp for comparisons
  Dangerous comparisons:
  - from < to (NodeElevator.sol#456)
  - block.timestamp < (dep.time + (packageList[dep.package].invest_days * day_secs)) (NodeElevator.sol#463-464)
  - block.timestamp > time_end (NodeElevator.sol#454)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

NodeElevator.unstakeBulk(uint256,uint256) (NodeElevator.sol#253-270) has costly operations inside a loop:
  - total_active_invested -= dep.amount (NodeElevator.sol#265)
NodeElevator.unstakeBulk(uint256,uint256) (NodeElevator.sol#253-270) has costly operations inside a loop:
  - totalUsers -- (NodeElevator.sol#266)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

```
SafeMath.add(uint256,uint256) (NodeElevator.sol#19-24) is never used and should be removed
SafeMath.mod(uint256,uint256) (NodeElevator.sol#137-139) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (NodeElevator.sol#153-160) is never used and should be removed
SafeMath.sub(uint256,uint256) (NodeElevator.sol#36-38) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (NodeElevator.sol#50-59) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version^0.8.0 (NodeElevator.sol#6) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Parameter NodeElevator.unstakeSingle(address)._user (NodeElevator.sol#272) is not in mixedCase
Parameter NodeElevator.updatePackages(uint40,uint256,uint256)._invest_days (NodeElevator.sol#288) is not in mixedCase
Parameter NodeElevator.updatePackages(uint40,uint256,uint256)._dailypercentage (NodeElevator.sol#289) is not in mixedCase
Parameter NodeElevator.deposit(uint16)._package (NodeElevator.sol#296) is not in mixedCase
Parameter NodeElevator.settleSingle(address,uint256,uint256)._addr (NodeElevator.sol#376) is not in mixedCase
Parameter NodeElevator.settleSingle(address,uint256,uint256)._at (NodeElevator.sol#377) is not in mixedCase
Parameter NodeElevator.revenueOf(address,uint256)._addr (NodeElevator.sol#404) is not in mixedCase
Parameter NodeElevator.revenueOf(address,uint256)._at (NodeElevator.sol#404) is not in mixedCase
Parameter NodeElevator.withdrawableRevenueOf(address)._addr (NodeElevator.sol#437) is not in mixedCase
Parameter NodeElevator.getUserInfo(address)._addr (NodeElevator.sol#477) is not in mixedCase
Variable NodeElevator.total_withdrawn (NodeElevator.sol#168) is not in mixedCase
Variable NodeElevator.total_invested (NodeElevator.sol#169) is not in mixedCase
Variable NodeElevator.total_active_invested (NodeElevator.sol#170) is not in mixedCase
Constant NodeElevator.day_secs (NodeElevator.sol#198) is not in UPPER_CASE_WITH_UNDERSCORES
Constant NodeElevator.percent_divide_by (NodeElevator.sol#201) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Reentrancy in NodeElevator.unstake(uint256) (NodeElevator.sol#344-373):
  External calls:
  - address(msg.sender).transfer(withdrawable) (NodeElevator.sol#361)
  State variables written after the call(s):
  - totalUsers -- (NodeElevator.sol#371)
  - total_active_invested -= dep.amount (NodeElevator.sol#367)
  - dep.isUnstaked = true (NodeElevator.sol#364)
  - dep.withdrawn += (withdrawable - dep.amount) (NodeElevator.sol#365)
  - users[msg.sender].invested -= dep.amount (NodeElevator.sol#366)
  Event emitted after the call(s):
  - NewWithdrawn(msg.sender,withdrawable) (NodeElevator.sol#362)
  - Unstaked(msg.sender,dep.amount) (NodeElevator.sol#368)
Reentrancy in NodeElevator.unstakeBulk(uint256,uint256) (NodeElevator.sol#253-270):
  External calls:
  - address(_user).transfer(users[_user].invested) (NodeElevator.sol#263)
  State variables written after the call(s):
  - totalUsers -- (NodeElevator.sol#266)
  - total_active_invested -= dep.amount (NodeElevator.sol#265)
  - dep.isUnstaked = true (NodeElevator.sol#262)
  - users[_user].invested = 0 (NodeElevator.sol#264)
Reentrancy in NodeElevator.unstakeSingle(address) (NodeElevator.sol#272-285):
  External calls:
  - address(_user).transfer(users[_user].invested) (NodeElevator.sol#279)
  State variables written after the call(s):
  - totalUsers -- (NodeElevator.sol#282)
  - total_active_invested -= dep.amount (NodeElevator.sol#281)
  - users[_user].invested = 0 (NodeElevator.sol#280)
Reentrancy in NodeElevator.withdraw() (NodeElevator.sol#331-340):
  External calls:
  - address(msg.sender).transfer(amount) (NodeElevator.sol#338)
  Event emitted after the call(s):
  - NewWithdrawn(msg.sender,amount) (NodeElevator.sol#339)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
```

```
NodeElevator.slitherConstructorConstantVariables() (NodeElevator.sol#163-528) uses literals with too many digits:
  - percent_divide_by = 100000 (NodeElevator.sol#201)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
NodeElevator.maxStake (NodeElevator.sol#204) should be constant
NodeElevator.minStake (NodeElevator.sol#203) should be constant
NodeElevator.unstakeFees (NodeElevator.sol#202) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
NodeElevator.ownerAddress (NodeElevator.sol#166) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
NodeElevator.sol analyzed (2 contracts with 84 detectors), 40 result(s) found
```

Solidity Static Analysis

HoldElevator.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in HoldElevator.withdraw(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 402:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 503:48:

Gas costs:

Gas requirement of function HoldElevator.unstakeBulk is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 262:4:

Gas costs:

Gas requirement of function HoldElevator.unstakeSingle is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 289:4:

Gas costs:

Gas requirement of function HoldElevator.revenueOf is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 413:4:

Gas costs:

Gas requirement of function `HoldElevator.withdrawableRevenueOf` is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 446:4:

This on local calls:

Use of "this" for local functions: Never use "this" to call functions in the same contract, it only consumes more gas than normal local calls.

[more](#)

Pos: 511:26:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

[more](#)

Pos: 505:8:

Constant/View/Pure functions:

`HoldElevator.withdrawableRevenueOf(address)` : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 446:4:

Similar variable names:

`HoldElevator.unstakeBulk(uint256,uint256)` : Variables have very similar names "user" and "users". Note: Modifiers are currently not considered by this static analysis.

Pos: 276:24:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 406:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 467:25:

NodeElevator.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in NodeElevator.unstake(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

[more](#)

Pos: 344:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

[more](#)

Pos: 491:48:

Gas costs:

Gas requirement of function NodeElevator.unstakeBulk is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 253:4:

Gas costs:

Gas requirement of function NodeElevator.revenueOf is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 404:4:

Gas costs:

Gas requirement of function NodeElevator.withdrawableRevenueOf is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 437:4:

Gas costs:

Gas requirement of function NodeElevator.getUserInfo is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 477:4:

This on local calls:

Use of "this" for local functions: Never use "this" to call functions in the same contract, it only consumes more gas than normal local calls.

[more](#)

Pos: 499:26:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

[more](#)

Pos: 493:8:

Similar variable names:

NodeElevator.unstakeBulk(uint256,uint256) : Variables have very similar names "users" and "_user". Note: Modifiers are currently not considered by this static analysis.

Pos: 257:8:

Similar variable names:

NodeElevator.unstakeSingle(address) : Variables have very similar names "users" and "_user". Note: Modifiers are currently not considered by this static analysis.

Pos: 275:38:

Similar variable names:

NodeElevator.getUserInfo(address) : Variables have very similar names "user" and "users". Note: Modifiers are currently not considered by this static analysis.

Pos: 494:34:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

[more](#)

Pos: 346:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 458:25:

Solhint Linter

HoldElevator.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:5
Error message for require is too long
Pos: 9:79
Explicitly mark visibility of state
Pos: 5:166
Variable name must be in mixedCase
Pos: 5:168
Variable name must be in mixedCase
Pos: 5:169
Variable name must be in mixedCase
Pos: 5:170
Variable name must be in mixedCase
Pos: 9:177
Variable name must be in mixedCase
Pos: 9:191
Explicitly mark visibility of state
Pos: 5:196
Constant name must be in capitalized SNAKE_CASE
Pos: 5:196
Explicitly mark visibility of state
Pos: 5:199
Constant name must be in capitalized SNAKE_CASE
Pos: 5:199
Variable name must be in mixedCase
Pos: 60:212
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:216
Code contains empty blocks
Pos: 31:224
Variable name must be in mixedCase
Pos: 9:244
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 25:275
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 25:276
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 25:299
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 25:300
Avoid making time-based decisions in your business logic
Pos: 30:329
Avoid making time-based decisions in your business logic
Pos: 52:343
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

```
Avoid making time-based decisions in your business logic
Pos: 17:348
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 9:363
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 9:364
Variable name must be in mixedCase
Pos: 17:379
Variable name must be in mixedCase
Pos: 17:381
Variable name must be in mixedCase
Pos: 17:422
Variable name must be in mixedCase
Pos: 17:424
Variable name must be in mixedCase
Pos: 17:454
Variable name must be in mixedCase
Pos: 17:456
Avoid making time-based decisions in your business logic
Pos: 29:462
Avoid making time-based decisions in your business logic
Pos: 76:462
Avoid making time-based decisions in your business logic
Pos: 25:471
Avoid making time-based decisions in your business logic
Pos: 37:481
Variable name must be in mixedCase
Pos: 13:492
Variable name must be in mixedCase
Pos: 13:496
Variable name must be in mixedCase
Pos: 13:497
Avoid making time-based decisions in your business logic
Pos: 49:502
Variable name must be in mixedCase
Pos: 13:507
Variable name must be in mixedCase
Pos: 13:529
```

NodeElevator.sol

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:5
Error message for require is too long
Pos: 9:79
Explicitly mark visibility of state
Pos: 5:165
Variable name must be in mixedCase
Pos: 5:167
Variable name must be in mixedCase
Pos: 5:168
Variable name must be in mixedCase
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

```
Pos: 5:169
Explicitly mark visibility of state
Pos: 5:170
Variable name must be in mixedCase
Pos: 9:177
Variable name must be in mixedCase
Pos: 9:191
Explicitly mark visibility of state
Pos: 5:197
Constant name must be in capitalized SNAKE_CASE
Pos: 5:197
Explicitly mark visibility of state
Pos: 5:200
Constant name must be in capitalized SNAKE_CASE
Pos: 5:200
Variable name must be in mixedCase
Pos: 60:212
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:216
Code contains empty blocks
Pos: 32:221
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 21:263
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 21:264
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 21:279
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 21:280
Variable name must be in mixedCase
Pos: 9:287
Avoid making time-based decisions in your business logic
Pos: 30:319
Avoid making time-based decisions in your business logic
Pos: 52:346
Avoid making time-based decisions in your business logic
Pos: 17:351
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 9:365
Possible reentrancy vulnerabilities. Avoid state changes after
transfer.
Pos: 9:366
Variable name must be in mixedCase
Pos: 13:382
Variable name must be in mixedCase
Pos: 13:384
Variable name must be in mixedCase
Pos: 17:413
Variable name must be in mixedCase
Pos: 17:415
Variable name must be in mixedCase
Pos: 17:445
Variable name must be in mixedCase
```

This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io

```
Pos: 17:447
Avoid making time-based decisions in your business logic
Pos: 29:453
Avoid making time-based decisions in your business logic
Pos: 76:453
Avoid making time-based decisions in your business logic
Pos: 25:462
Avoid making time-based decisions in your business logic
Pos: 37:472
Variable name must be in mixedCase
Pos: 13:480
Variable name must be in mixedCase
Pos: 13:484
Variable name must be in mixedCase
Pos: 13:485
Avoid making time-based decisions in your business logic
Pos: 49:490
Variable name must be in mixedCase
Pos: 13:495
Variable name must be in mixedCase
Pos: 13:517
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: audit@EtherAuthority.io