



Audit Report

Ethernity Chain

Fixed Staking-v2

Apr 2023



Content

- Disclaimer
- Audit Goals
- Audit Details
- Security Level Reference
- Findings
- Additional Details
- Concluding Remarks



Disclaimer

This disclaimer is to inform you that the report you are reading has been prepared by Mantisec Labs for informational purposes only and should not be considered as investment advice. It is important to conduct your own independent investigation before making any decisions based on the information contained in the report. The report is provided "as is" without any warranties or conditions of any kind and Mantisec Labs excludes all representations, warranties, conditions, and other terms. Additionally, Mantisec Labs assumes no liability or responsibility for any kind of loss or damage that may result from the use of this report.

It is important to note that the analysis in the report is limited to the security of the smart contracts only and no applications or operations were reviewed. The report contains proprietary information, and Mantisec Labs holds the copyright to the text, images, photographs, and other content. If you choose to share or use any part of the report, you must provide a direct link to the original document and credit Mantisec Labs as the author.

By reading this report, you are accepting the terms of this disclaimer. If you do not agree with these terms, it is advisable to discontinue reading the report and delete any copies in your possession.



Audit Goals

Mantise Labs was commissioned by the Ethernity team to perform an audit of their smart contract. The audit was conducted in the month of **April 2023**.

The purpose of this audit was to achieve the following:

- i. Identify potential security issues within the smart contract
- ii. Formally check the logic behind the given smart contract

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



Audit Details

- Project Name: Ethernity Chain
- Contract Name: [FixedStaking.sol](#)
- Languages: Solidity(Smart contract)
- Github link: <https://github.com/ethernitychain/fixed-staking-v2/blob/main/contracts/FixedStaking.sol>
- Platforms and Tools: Remix IDE, Solhint, VScode, Contract Library, Slither



Security Level Reference

Every issue in this report were assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	0	0	0
Closed	2	2	4



Findings

High Severity Issues

1. Function `setPenalty` incorrectly checks `penalty` variable instead of `_penalty` argument

Explanation:

The `penalty` variable is being checked in the `require` statements instead of the `_penalty` argument. This means that the function will always check the current value of `penalty` instead of the new value that is being passed in as an argument.

2. Missing logic for adding subtracted penalty amount to the reward pool in `computeReward` function

Explanation:

When a penalty is applied to a stake, the deducted amount should be added back to the rewards pool. This means that if a user cancels their stake before it matures, they will receive a reduced reward due to the penalty, but the deducted amount will be added back to the rewards pool to be distributed to other stakers. This helps to ensure that the rewards pool stays properly funded.

Medium Severity Issues

1. Confusion in Contract Requirement and Implementation Logic for Penalty Calculation in `computeReward` function

Explanation:

There appears to be some confusion in the contract requirements and the logic behind the code. While the current implementation is correct for deducting 70% when the penalty is set to 30, it seems illogical that a penalty of 30% would mean deducting 30% from the reward. Instead, it should deduct 70% from the reward, leaving only 30%. To implement this, the correct formula for subtracting the penalty from the reward would be "reward = reward - ((reward * penalty) / 100)."

2. Hardcoded value for stake claim delay

Explanation:

In the `claim()` function of the contract, there is a hardcoded delay of 5 days for claiming a cancelled stake. This delay is used to ensure that users cannot claim their stake immediately after cancelling it, and must wait for a period of time before claiming.

However, this delay is not adjustable and is not in line with the contract's requirement for a dynamic penalty duration.

To address this issue, the delay should be replaced with the `penaltyDuration` variable that can be set by the contract owner. This would allow the owner to adjust the delay period as needed, without having to change the contract's code.

Low Severity Issues

1. Function `setPenalty` allows a penalty value of zero, contrary to requirement.

Explanation:

The `setPenalty` function allows a penalty value of zero to be set, which is not allowed as per the requirements. The first `require` statement should check that `_penalty` is greater than zero instead of greater than or equal to zero. This issue can be fixed by updating the `require` statement accordingly.

2. Lack of Validation in `setStakeLimits` Function Parameters

Explanation:

The `setStakeLimits` function should include additional validation on the parameters to ensure that `_minStakeAmount` is greater than zero and `_maxStakeAmount` is greater than `_minStakeAmount` or some upper bound. Without these validations, the function may set stake limits that do not meet the contract's requirements.

3. Ambiguous Variable Name Usage in `Stake` Struct Initialization.

Explanation:

The **restake** function allows a user to re-stake their previous stake, with the addition of any earned rewards. However, the variable **amount** is used to store both the original stake amount and the newly computed reward, which can be confusing. It would be better to use a different variable name for the new amount being staked.

4. Missing Functionality for Changing Stake Type in **restake** Function

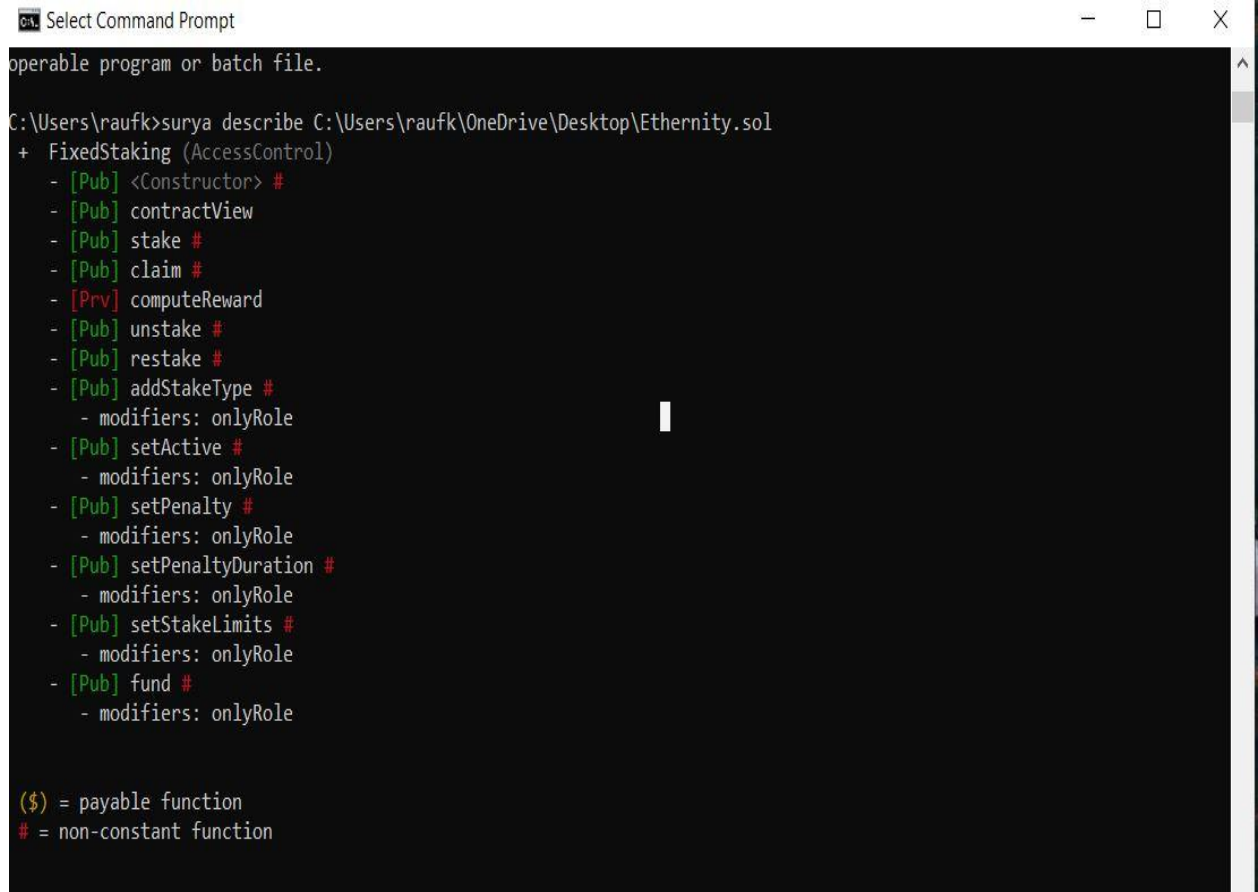
Explanation:

The issue is that the **restake** function does not allow the user to change the stake type when restaking, which limits the flexibility of the contract. Currently, the function automatically restakes the same stake type as the original stake. If a user wants to restake with a different stake type, they would need to create a new stake instead of using the **restake** function.

This limitation can be problematic if a user wants to take advantage of a different stake type that has become available or if they want to change their investment strategy. A more flexible solution would allow users to restake with any available stake type.

Additional Details

1. Contract Description

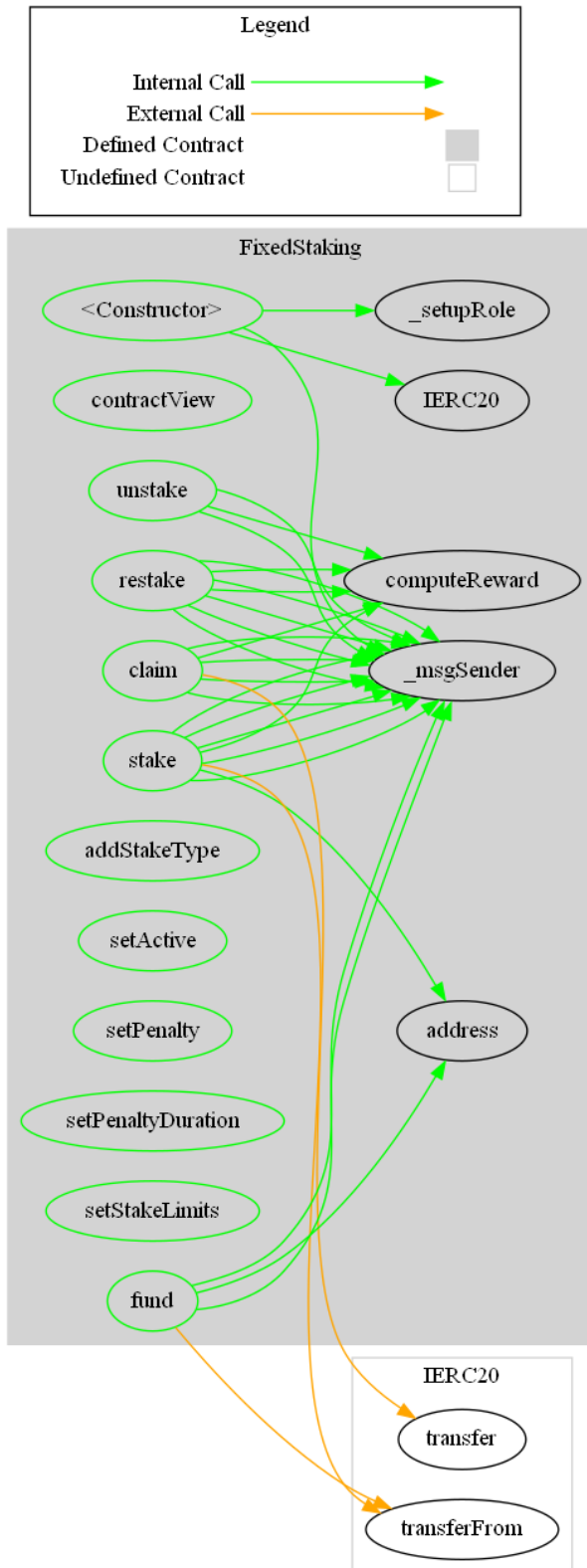


```
Select Command Prompt
operable program or batch file.

C:\Users\raufk>surya describe C:\Users\raufk\OneDrive\Desktop\Ethernity.sol
+ FixedStaking (AccessControl)
  - [Pub] <Constructor> #
  - [Pub] contractView
  - [Pub] stake #
  - [Pub] claim #
  - [Prv] computeReward
  - [Pub] unstake #
  - [Pub] restake #
  - [Pub] addStakeType #
    - modifiers: onlyRole
  - [Pub] setActive #
    - modifiers: onlyRole
  - [Pub] setPenalty #
    - modifiers: onlyRole
  - [Pub] setPenaltyDuration #
    - modifiers: onlyRole
  - [Pub] setStakeLimits #
    - modifiers: onlyRole
  - [Pub] fund #
    - modifiers: onlyRole

($) = payable function
# = non-constant function
```

2. Call Graph





Concluding Remarks

While conducting the audit of the fixed staking smart contract, it was observed that the contracts contained High, Medium, and Low severity issues.

Our auditors confirmed that High, Medium, and Low severity issues are now resolved by the developers.