Audit Interim Report

This is an interim Smart Contract Audit Report that is executed for proper communication between Saif Sghaier and its clients. This is not to be considered a final report.

Project Name - *Being*
Project Platform - *EVM*
Project Language - *Solidity*

Project Contract Link -*<Enter Contract link here>*

Project CodeBase -
https://docs.google.com/document/u/0/d/1-Yc1LPIH8ATB2gmepbuiXVcIQUWctRYJ8PXlCbjG4FU/mobilebasic?pli=1
Project Commit - *<Enter Commit Hash for codebase here>*

File Details

<Enter Name of File and Give It A File ID>
*<File ID **Naming Convention** - File ID will contain 3 letters. The first two letters will be initials from the project name, the third letter will be the initial of the file name. If two or more files contain same initial, then a fourth letter might be added to distinguish between them>*
*<File ID **Example** -*
Project name - Lightning Works
File names - LW0-Contract.sol, LW0-Minter.sol, LW0-Simple.sol
File IDs - LWC, LWM, LWS
Issues IDs- LWC01, LWC02, LWC03 etc.>

File ID	File Name
BEING	Being-being.sol

Audit Details

Report Submission Date - 09/12/2024
Result - Not Passed

Findings Details

Severity	Number Of Issues	Percentage
Critical	7	63%
High	1	9%
Medium	0	0%
Low	2	18%
Informational	1	10%

Finding Summary

*<Issues **Status** Details -*

***Reported** - When Issue is first reported.*

***Acknowledged** - If client has seen the issues but not taken any action*

***Resolved** - If client has seen the issue and fixed it>*

Issue ID	Type	Line	Severity	Status
BEING-01	deploymentTime ≠ DEPLOYMENT_TIMESTAMP	45	Critical Severity	Resolved
BEING-02	Wrong Reward Calculation	95-106	Critical Severity	Reported
BEING-03	totalStakedBalance has to be a global variable	70 / 86 / 121	Critical Severity	Resolved
BEING-04	lastStakeTime Uses block.number instead of block.timestamp	68	Critical Severity	Resolved
BEING-05	Unused local Variable	76	Critical Severity	Resolved
BEING-06	Staking and Unstaking Logic	75-93	Critical Severity	Reported
BEING-07	functions with same signature	-	Critical Severity	Resolved
BEING-08	getTotalStaked() does not return contract balance	78	High Severity	Reported
BEING-09	Centralization Risk	-	Low Severity	Acknowledged
BEING-10	Minimum Stake Amount	17	Low Severity	Acknowledged
BEING-11	floating pragma & Old solidity version	2	Informational	Resolved

Issue ID - *BEING-01*

Type - deploymentTime ≠ DEPLOYMENT_TIMESTAMP
Severity - Critical Severity
File - being.sol
Line - 45
Status - Resolved

Description - in getCurrentRewardRate(), DEPLOYMENT_TIMESTAMP is used to calculate timeElapsed but the correct variable name is deploymentTime which is a global variable initialized in the constructor.

Remediation - changed the global variable name to DEPLOYMENT_TIMESTAMP in the constructor too. Also make it IMMUTABLE since it does not change after deployment.

SnapShot -

```
function getCurrentRewardRate() public view returns (uint256) {
        uint256 timeElapsed = block.timestamp - DEPLOYMENT_TIMESTAMP;

        if (timeElapsed < 365 days) return FIRST_YEAR_RATE;

        if (timeElapsed < 2 * 365 days) return SECOND_YEAR_RATE;

        return THIRD_YEAR_RATE;
    }
```

Issue ID - *BEING-02*

Type - Wrong Reward Calculation
Severity - Critical Severity
File - being.sol
Line - 95-106
Status - Reported

Description - Rewards are calculated using integer division, which can lead to precision loss, especially for small amounts. Additionally, the calculation doesn't account for compounding over multiple staking periods.

Remediation - Consider using a higher precision unit for calculations (e.g., using a multiplier for decimals). Implement a mechanism to account for compounding if desired.
Here is the right way to calculate rewards in the code snippet below:
SnapShot -

```
function calculateRewards(address staker) public view returns (uint256) {
    uint256 stakedAmount = stakedBalance[staker];
    if (stakedAmount == 0) return 0;

    uint256 startTime = lastStakeTime[staker];
    uint256 endTime = block.timestamp;
    uint256 totalRewards = 0;

    // Calculate rewards for each year period separately
    uint256 firstYearEnd = DEPLOYMENT_TIMESTAMP + YEAR_DURATION;
    uint256 secondYearEnd = firstYearEnd + YEAR_DURATION;

    // First year rewards
    if (startTime < firstYearEnd) {
        uint256 endPeriod = min(endTime, firstYearEnd);
        uint256 duration = endPeriod - startTime;
        totalRewards += (stakedAmount * FIRST_YEAR_RATE * duration) / (YEAR_DURATION * 100);
        startTime = endPeriod;
    }

    // Second year rewards
    if (startTime < secondYearEnd && endTime > firstYearEnd) {
        uint256 endPeriod = min(endTime, secondYearEnd);
        uint256 duration = endPeriod - startTime;
        totalRewards += (stakedAmount * SECOND_YEAR_RATE * duration) / (YEAR_DURATION * 100);
        startTime = endPeriod;
    }

    // Third year and beyond rewards
    if (endTime > secondYearEnd) {
        uint256 duration = endTime - startTime;
        totalRewards += (stakedAmount * THIRD_YEAR_RATE * duration) / (YEAR_DURATION * 100);
    }

    return totalRewards;
}

// Helper function to get minimum of two numbers
function min(uint256 a, uint256 b) private pure returns (uint256) {
    return a < b ? a : b;
}
```

Issue ID - *BEING-03*

Type - totalStakedBalance has to be a global variable
Severity - Critical Severity
File - being.sol
Line - 70 / 86 / 121
Status - Resolved

Description - totalStackedBalance is used to keep track of the total stacked balance in the contract but it is not declared at all which makes the contract uncompilable.

Remediation - Declare the totalStakedBalance as a global variable.

SnapShot -

```
totalStakedBalance += amount;
```

```
totalStakedBalance -= stakedBalance[msg.sender];
```

```
function getTotalStaked() external view returns (uint256) {
    return totalStakedBalance; // Use the declared variable
}
```

Issue ID - *BEING-04*

Type - lastStakeTime Uses block.number instead of block.timestamp
Severity - Critical Severity
File - being.sol
Line - 68
Status - Resolved

Description - lastStakeTime uses block.number to keep track of the last time a user staked but it should be block.timestamp instead. This lead to major issue with the contract logic

Remediation - Use block.timestamp instead of block.number

SnapShot -

```
function stakeTokens(uint256 amount) external nonReentrant {
    require(
        amount >= MINIMUM_STAKE_AMOUNT,
        "Stake amount must be at least the minimum."
    );

    require(
        balanceOf(msg.sender) >= amount,
        "Insufficient balance to stake."
    );

    _transfer(msg.sender, address(this), amount); // Lock tokens in contract

    stakedBalance[msg.sender] += amount;

    lastStakeTime[msg.sender] = block.number;

    totalStakedBalance += amount;

    emit TokensStaked(msg.sender, amount);
}
```

Issue ID - *BEING-05*

Type - Unused local Variable
Severity - Critical Severity
File - being.sol
Line - 76
Status - Resolved

Description - uint256 stakedAmount = stakedBalance[msg.sender]; is used to get the staked amount of a certain staker in unstakeTokens() but it is not used in the function, also stakedBalance[msg.sender] is set to zero before transferring the funds to the staker which will make the staker lose his staked funds.

Remediation - Use stackedAmount in the function after setting the stakedBalance[msg.sender] to 0.

SnapShot -

```
function unstakeTokens() external nonReentrant {
    uint256 stakedAmount = stakedBalance[msg.sender];

    require(stakedBalance[msg.sender] > 0, "No tokens staked");

    uint256 rewards = calculateRewards(msg.sender);

    stakedBalance[msg.sender] = 0;

    lastStakeTime[msg.sender] = 0;

    totalStakedBalance -= stakedBalance[msg.sender];

    _mint(msg.sender, rewards); // Mint rewards to staker

    _transfer(address(this), msg.sender, stakedBalance[msg.sender]);

    emit TokensUnstaked(msg.sender, stakedBalance[msg.sender] + rewards);
}
```

Issue ID - *BEING-06*

Type - Staking and Unstaking Logic
Severity - Critical Severity
File - being.sol
Line - 75-93
Status - Reported

Description - The contract allows unstaking of all tokens at once without any penalties or lock periods. This could lead to potential issues with tokenomics, such as users staking just before a reward period ends and unstaking immediately after.

Remediation - Implement a lock period or penalties for early unstaking to encourage longer-term staking. This can help stabilize the token economy.

SnapShot -

```
function unstakeTokens() external nonReentrant {
    uint256 stakedAmount = stakedBalance[msg.sender];

    require(stakedAmount > 0, "No tokens staked");

    uint256 rewards = calculateRewards(msg.sender);

    stakedBalance[msg.sender] = 0;

    lastStakeTime[msg.sender] = 0;

    totalStakedBalance -= stakedAmount;

    _mint(msg.sender, rewards); // Mint rewards to staker

    _transfer(address(this), msg.sender, stakedAmount);

    emit TokensUnstaked(msg.sender, stakedAmount + rewards);
}
```

Issue ID - *BEING-07*

Type - functions with same signature
Severity - Critical Severity
File - being.sol
Line - -
Status - Resolved

Description - By the end of the contract there are two functions that have the same signature:
function getTotalStaked() external view returns (uint256) {
    return totalSupply(); // Replace with the inherited or correct variable/method.
}

  function getTotalStaked() external view returns (uint256) {
    return totalStakedBalance; // Use the declared variable
}

Making the contract uncompilable

Remediation - remove the one that returns totalSupply(); because total supply is already a public variable from ERC20.sol

SnapShot -

```
function getTotalStaked() external view returns (uint256) {

    return totalSupply(); // Replace with the inherited or correct variable/method.

}

   function getTotalStaked() external view returns (uint256) {

    return totalStakedBalance; // Use the declared variable

}
```

Issue ID - *BEING-08*

Type - getTotalStaked() does not return contract balance
Severity - High Severity
File - being.sol
Line - 78
Status - Reported

Description - getTotalStaked() does not return the contract balance, instead it returns how much ETH is inside the contract.

Remediation - To get the real balance of the contract change it to this: Return balanceOf(address(this));

SnapShot -

```
function getTotalStaked() external view returns (uint256) {
        return address(this).balance;
    }
```

Issue ID - *BEING-09*

Type - Centralization risk
Severity - Low Severity
File - being.sol
Line - -
Status - Acknowledged

Description - The Owner of the contract holds all the privileges. It is considered a bad practice and can lead to loss of funds or losing control of the protocol if the owner address is compromised.

Remediation - Consider adding more roles (admins) or using a multisignature.

SnapShot -
No snapshot required.

Issue ID - *BEING-10*

Type - Minimum Stake Amount
Severity - Low Severity
File - being.sol
Line - 17
Status - Acknowledged

Description - The minimum stake amount is hardcoded, which reduces flexibility for future changes.

Remediation - Allow the owner to update the minimum stake amount through a function, ensuring it can adapt to future requirements.

SnapShot -

```
1   uint256 public constant MINIMUM_STAKE_AMOUNT = 1000;
```

Issue ID - *BEING-11*

Type - floating pragma & Old solidity version
Severity - Informational
File - being.sol
Line - 3
Status - Resolved

Description - the contract is using an old solidity version, plus is it unlocked meaning it can be compiled with any version from 0.8.0 to the latest version. It is not considered a best practice.

Remediation - use the latest stable solidity version and lock it to a fixed version. E.g: 0.8.24

SnapShot -

```
1   pragma solidity ^0.8.0;
```