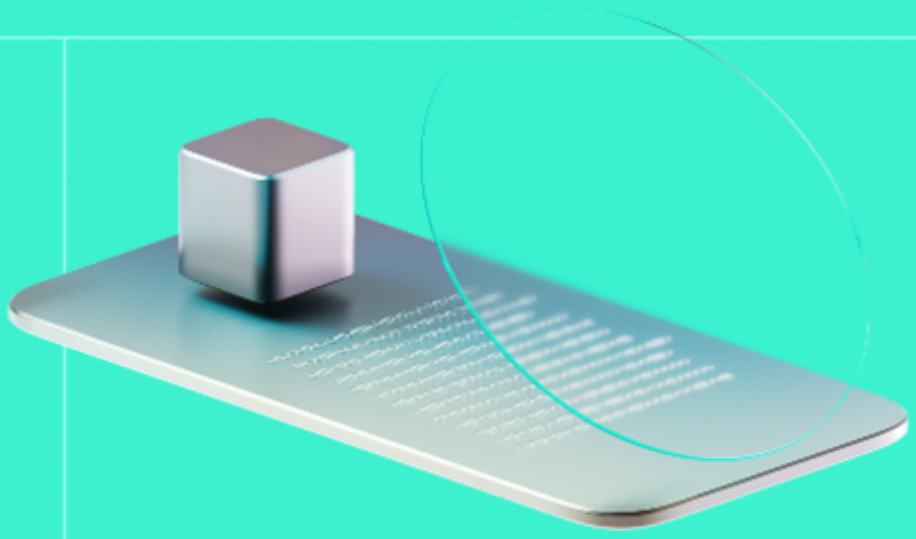
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Wow Earn

Date: 12/01/2024

We thank Wow Earn for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Wow Earn aims to build a seamless connection between users and the blockchain world web3 Platform by providing healthy and sustainable blockchain application.

Platform: EVM

Language: Solidity

Tags: Vesting, reward distribution

Timeline: 11/12/2023 - 12/01/2024

Methodology: https://hackenio.cc/sc_methodology

Last Review Scope

Repository	https://github.com/ullaniubility/ulla-contract
Commit	ab46260719de5ca70cee6dad21336bc81b274ae9

Audit Summary

10/10	8/10	76,92%	8/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.4/10

The system users should acknowledge all the risks summed up in the risks section of the report

6	3	3	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	6

Vulnerability	Status
F-2023-0147 - Missing input validation for key parameters	Accepted
F-2023-0154 - The totalCumulative state variable can be calculated only once	Accepted
F-2023-0157 - The amount() function can return inaccurate value	Accepted
F-2023-0143 - The lastTime() function can revert due to arithmetic underflow	Fixed
F-2023-0153 - The start() function lacks input validation	Fixed
F-2023-0155 - Lack of two-step ownership transfer	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Wow Earn
Audited By	Grzegorz Trawinski
Approved By	Przemyslaw Swiatowiec, Ataberk Yavuzer
Website	https://www.wowearn.com/
Changelog	12/01/2024 - Final Report

Table to Contents

System Overview

The mineWOW contract is a centralised solution where privileged account can add and configure users that will have rewards accounted over time. Rewards are accounted in the form of native assets. Users can obtain rewards proportionally, issued over time.

Privileged roles

The mineWOW's owner can:

- Register or re-register new user.
- Distribute reward to the user.
- Set essential state variables, e.g. unlock time.
- Transfer ownership to another account.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

Code quality

The total Code Quality score is **8** out of **10**.

- No development environment configuration was provided.
- Few observations were reported regarding redundant code items.
- Few observations were reported regarding Gas extensive usage.
- Few instances of inaccurate naming of function and variables where identified.

Test coverage

Code coverage of the project is **76,92%** (branch coverage).

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **6** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **9.4**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The mineWOW contract is heavily centralised solution. Users have no control over protocol configuration. The rewards are distributed only to the users configured by the privileged account.

Findings

Vulnerability Details

F-2023-0143 - The lastTime() function can revert due to arithmetic underflow - Low

Description:

The `lastTime()` function attempts to return the difference between the constant value representing day in seconds (`86400`) and difference between current `block.timestamp` and user's `startTime`, However, this function reverts with `Arithmetic over/underflow` error message whenever `86401` seconds passes since user's `startTime` was set.

```
function lastTime(address addr) public view returns (uint256) {
        return 86400 - (getCurrentTime() - userInfo[addr].startTime);
    }
```

This vulnerability makes this particular function unusable for future usage.

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 3/5

Recommendations

Recommendation: It is recommended to review the purpose of this function and fix the implementation so it returns valid value every time.

Remediation (revised commit: 5f11f2) : The aforementioned function does not revert anymore.

F-2023-0147 - Missing input validation for key parameters - Low

Description: The contract's constructor is missing input validation for key parameters: `intervalCount` and `ratioCount`. Both parameters participate in the reward accounting and distribution processes. Setting these variables to 0 can cause no reward or underestimated reward accounted. Additionally, the both parameters lack upper bound check. Thus, it can be set to any value, resulting in possible inaccurate reward accounting.

```
constructor(uint256 _intervalCount,uint256 _ratioCount) {
    owner = payable(msg.sender);
    intervalCount = _intervalCount;
    ratioCount = _ratioCount;
}
```

Status: Accepted

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation: It is recommended to implement input validation for key parameters in accordance to the solution business rules.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0153 - The start() function lacks input validation - Low

Description: The `start()` function register or re-register new user in the contract. However, it does not implement any input validation. The `addr` parameter could be verified against 0×0 address, so it would not register invalid user. The `speed` parameter could be verified against 0 value. This particular parameter is used in `calculateReward()` to calculate the reward, and whenever it is set to 0, the reward is 0 as well. Additionally, the speed parameter lacks upper bound check. Thus, it can be set to any value, resulting in possible enormous reward accounting.

```
function start(address addr, uint256 speed) public returns (uint256){
        uint256  previousRewards = register(addr, speed);
        emit startLog(addr, speed, previousRewards);
        return  previousRewards;
    }
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation: It is recommended to implement input validation in accordance to the solution business rules.

Remediation (revised commit: 5f11f2) : The `addr` parameter has now input validation implemented against zero address. The `speed` parameter can be set to 0 value.

F-2023-0154 - The totalCumulative state variable can be calculated only once - Low

Description:

The `sumRewardDebt()` function calculates the `totalCumulative` state variable basing on each registered user `rewardDebt`. Also, whenever this state variable is calculated, it cannot be updated anymore. However, it was noticed that user's `rewardDebt` is only updated upon calling the `getCount()`, `transfer()`, `transferOwner()` functions for each user separately, assuming that `locktime` was updated previously. This design can result in situation when the `totalCumulative` state variable has incorrect value calculated, as some user had not `rewardDebt` updated in prior.

```
function sumRewardDebt() public  returns (uint256) {
     require(owner == msg.sender, "Query failed.");
     require(locktime > 0, "Query failed");
     if(totalCumulative == 0){
        for (uint256 i = 0; i < addressList.length; i++) {
             address userAddress = addressList[i];
             UserInfo memory info = userInfo[userAddress];
             totalCumulative += info.rewardDebt;
         }
     }
     return totalCumulative;
 }
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Classification

Severity: Low

Impact: 2/5

Likelihood: 2/5

Recommendations

Recommendation: It is recommenced to update the `totalCumulative` state variable anytime the user's `totalRewardDebt` variables are updated.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0155 - Lack of two-step ownership transfer - Low

Description: The contract implements single-step ownership transfer functionality. Thus, accidental transfer of ownership to unverified and incorrect address may result in loss of ownership. In such a case, access to every function protected by the `owner` constraint check will be permanently lost.

```
function transferOwnership(address newOwner) public virtual  {
    require(owner == msg.sender, "Insufficient permissions.");
    require(newOwner != address(0), "Invalid address.");
    address oldOwner = owner;
    owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Classification

Severity: Low

Impact: 3/5

Likelihood: 2/5

Recommendations

Recommendation: It is recommended to implement a two-step ownership transfer pattern within the solution, such as proposed in OpenZepplin's Ownable2Step.

Remediation (revised commit: 5f11f2) : The `transferOwnership()` function is now removed from the code base.

F-2023-0157 - The amount() function can return inaccurate value - Low

Description:

The `amount()` function appears to be returning the accounted reward for particular user. To calculate reward it takes current time as input parameter. However, it does not take into account that `locktime` could be already set. Thus, for some period, until particular user updates the `totalRewardDebt` variable, this function can return inaccurate value.

```
function amount(address addr) public view returns (uint256) {
    if (userInfo[addr].totalRewardDebt == 0) {
        uint256 count = calculateReward(addr, getCurrentTime());
        return userInfo[addr].rewardDebt + count;
    } else {
        return userInfo[addr].totalRewardDebt;
    }
}
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Classification

Severity: Low

Recommendations

Recommendation:

It is recommended to fix the function implementation so it returns accurate value every time.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

Observation Details

F-2023-0140 - State variables default visibility - Info

Description: It was identified that multiple variables lack of visibility declared. Declaring state variables visibility explicitly support engineers to catch incorrect assumptions about variables accesses. Default variable visibility in such case is `internal`.

```
address owner;
uint baseQuota = 83333333;
uint locktime;
uint baseSpeed = 100000;
uint256 totalCumulative;
uint256 intervalCount;
uint256 ratioCount;
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Recommendations

Recommendation: It is recommended to explicitly declare each variable as `public`, `internal`, or `private`.
Remediation (revised commit: 5f11f2) : All variables now have visibility declared.

F-2023-0141 - Unused events - Info

Description: It was identified that multiple events are declared but not utilized in any of the solution's functionality. Having unused event declarations consumes additional Gas during the deployment.

```
event LogData(string message, uint256 data);
event countEvent(uint256 value);
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Recommendations

Recommendation: It is recommended to remove unused events.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0142 - Gas overconsumption in deposit mechanism - Info

Description: The solution uses `balances` array to track native tokens deposited. This array has only one record for current's contract address (`address(this)`). Such approach uses additional storage slots and consumes additional Gas. Such processing is being done within the `deposit()`function. Additionally, the `balances` array is not updated upon transferring tokens to the users.

```
mapping(address => uint256) public balances;
```

```
    function deposit() external payable {
        balances[address(this)] += msg.value;
    }
```

Status: Accepted

Recommendations

Recommendation: It is recommended to remove `balances` array, instead the contract's balance can be checked natively. Also, it is recommended to remove the `deposit()` function, and instead implement the native `receive()` function to enable native tokens processing.
Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0144 - Redundant SafeMath library in use - Info

Description:

Prior to Solidity version 0.8.0, arithmetic overflows were not handled natively by the language, and developers were encouraged to use the `SafeMath` library as a safeguard against such errors.

However, with the release of Solidity version 0.8.0, the language introduced new arithmetic overflow and underflow protection features that made the `SafeMath` library redundant if using Solc versions above 0.8.0.

Currently, the solution uses Solc version of 0.8.7.

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Recommendations

Recommendation:

It is recommended to remove the usage the `SafeMath` library for arithmetic operations, and Instead, leverage the native arithmetic overflow and underflow protection features.

Remediation (revised commit: 5f11f2) : The `SafeMath` library and its usage is now removed.

F-2023-0145 - The getCurrentTime() is redundant wrapper - Info

Description: The `getCurrentTime()` function appears to be redundant wrapper on `block.timestamp` global variable. Such global variable is always accessible and it does not require any additional getters. Current implementation leads to extensive Gas usage during both deployment and usage.

```
function getCurrentTime() public view returns (uint256 blockTime) {
    return block.timestamp;
}
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Recommendations

Recommendation: It is recommended to remove redundant wrapper and use global variable instead.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0148 - Native tokens transfer via the transfer() function - Info

Description:

The mineWOW's `transfer()` and `transferOwner()` functions use built-in `transfer()` function for transferring native tokens.

The `transfer()` function was commonly used in earlier versions of Solidity for its simplicity and automatic reentrancy protection. However, it was identified as potentially problematic due to its fixed gas limit of 2300.

The usage of `transfer()` function can lead to unintended function call revert when the receiving contract's `receive()` or `fallback()` functions require more than 2300 Gas for processing.

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Recommendations

Recommendation:

It is recommended to use built-in `call()` function instead of `transfer()` to transfer native assets. This method does not impose a gas limit, it provides greater flexibility and compatibility with contracts having more complex business logic upon receiving the native tokens. When working with then `call()` function ensure that its execution is successful by checking the returned boolean value. It is also recommended to fallow the Check-Effects-Interactions (CEI) pattern in every case to prevent reentrancy issues.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0149 - Lack of days time unit usage across solution - Info

Description:

Within the solution multiple instances of literal value representing seconds in day (`86400`) were identified. Also, within the `calculateReward()` function seconds in day in seconds are being calculated each function call (`24 * 60 * 60`). In Solidity language there is a suffix keyword that represents seconds in day unit time: `days`. As a result, current implementation leads to extensive Gas consumption.

```
function transferOwner() public {
[...]
        uint256 day = calculateDay(getCurrentTime(), locktime, 86400);
[...]
    }
```

```
function getCount(address payable from) public returns (uint256) {
[...]
        uint256 dayTime = calculateDay(getCurrentTime(), locktime, 86400);
[...]
    }
```

```
function getExtractCount(address from) public view returns (uint256) {
[...]
        uint256 dayTime = calculateDay(getCurrentTime(), locktime, 86400);
[...]
    }
```

```
function lastTime(address addr) public view returns (uint256) {
        return 86400 - (getCurrentTime() - userInfo[addr].startTime);
    }
```

```
    function calculateReward(
        address addr,
        uint256 time
    ) public view returns (uint256) {
        UserInfo memory newUser = userInfo[addr];
        uint256 count = 0;
        if (newUser.startTime > 0 && time > newUser.startTime) {
            uint256 timePassed = time - newUser.startTime;
            if (time - newUser.startTime > 24 * 60 * 60)
                timePassed = 24 * 60 * 60;
            count = timePassed.mul(newUser.speed).mul(baseQuota);
        } else {
            count = 0;
```

```
        }
        return count;
    }
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Recommendations

Recommendation: It is recommended to use `days` time unit in every aforementioned instance.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0150 - Redundant variable value assignments - Info

Description: Within the `calculateReward()` function the `count` variable is assigned twice to 0. In both cases it is redundant as by default variable of such type is assigned to 0. Thus, such approach leads to extensive Gas consumption.

```
function calculateReward(
        address addr,
        uint256 time
    ) public view returns (uint256) {
        UserInfo memory newUser = userInfo[addr];
        uint256 count = 0;
        if (newUser.startTime > 0 && time > newUser.startTime) {
            uint256 timePassed = time - newUser.startTime;
            if (time - newUser.startTime > 24 * 60 * 60)
                timePassed = 24 * 60 * 60;
            count = timePassed.mul(newUser.speed).mul(baseQuota);
        } else {
            count = 0;
        }
        return count;
    }
```

Status: Fixed

Recommendations

Recommendation: It is recommended to remove the assignments to default value.

Remediation (revised commit: 5f11f2) : The second assignment to default value is now removed.

F-2023-0151 - Misleading error messages - Info

Description: It was identified that error messages presented by the solution whenever assertion fails can be considered misleading. E.g. only within the `transferOwnership()` function the access control assertion reverts with `Insufficient permissions.` error message. In every other case it is a `failed` message. In other case, the same error message (`transfer failed.`) is provided for two distinct assertions. Such approach can be misleading for end users and it can increase time required for possible troubleshooting during emergency event.

```
    function transfer(address from) public {
        require(owner == msg.sender, "transfer failed.");
        require(locktime > 0, "transfer failed");
[...]
    }

    function transferOwner() public {
        require(locktime > 0, "transfer failed");
[...]
    }



    function unlock() public {
        require(owner == msg.sender, "Mining failed.");
        require(locktime == 0, "Mining has ended.");
        locktime = getCurrentTime();
    }

    function register(address addr, uint256 speed) private  returns (uint256) {
        require(owner == msg.sender, "Mining failed.");
        uint256 time = getCurrentTime();
        require(locktime == 0, "Mining has ended.");
[...]
    }

    function sumRewardDebt() public  returns (uint256) {
        require(owner == msg.sender, "Query failed.");
        require(locktime > 0, "Query failed");
[...]
    }



    function transferOwnership(address newOwner) public virtual  {
        require(owner == msg.sender, "Insufficient permissions.");
        require(newOwner != address(0), "Invalid address.");
```

```
[...]
    }
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Recommendations

Recommendation: It is recommended to refactor strings in assertions to provide accurate and meaningful errors messages.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0152 - Custom Errors in Solidity for Gas Efficiency - Info

Description:

Starting from Solidity version 0.8.4, the language introduced a feature known as *custom errors*. These custom errors provide a way for engineers to define more descriptive and semantically meaningful error conditions without relying on string messages. Prior to this version, engineers often used the `require` statement with string error messages to handle specific conditions or validations. However, every unique string used as a revert reason consumes additional gas, making transactions more expensive.

Custom errors, on the other hand, are identified by their name and the types of their parameters only, and they do not have the overhead of string storage. This means that, when using custom errors instead of `require` statements with string messages, the gas consumption can be significantly reduced, leading to more gas-efficient contracts.

The solution uses Solc version of **0.8.7**.

Example `require` statement with string error message:

```
function unlock() public {
    require(owner == msg.sender, "Mining failed.");
    require(locktime == 0, "Mining has ended.");
    locktime = getCurrentTime();
}
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Recommendations

Recommendation:

It is recommended to use custom errors instead of revert strings to reduce gas costs, especially during contract deployment. Custom errors can be defined using the error keyword and can include dynamic information.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0156 - Redundant condition within the register() function - Info

Description: Within the `register()` function there is a condition check that is redundant (`speed >= 0`). The `speed` variable is a value in a range between 0 and maximum value for `uint256` type, so this condition is always satisfied. Thus, such implementation leads to extensive Gas consumption.

```
function register(address addr, uint256 speed) private  returns (uint256) {
[...]
        if (speed >= 0) userInfo[addr].speed = speed;
        return reward;
    }
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Recommendations

Recommendation: It is recommended to remove redundant condition check.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0170 - Constant variables can be introduced - Info

Description: Within the solution, the `baseQuota` and `baseSpeed` variables appear to have constant values, however they are not declared as `constant`. Constant variables does not occupy storage slots, instead they increase the executable size. Thus, having constant variables may save some Gas.

```
uint baseQuota = 83333333;
uint baseSpeed = 100000;
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Recommendations

Recommendation: It is recommended to declare aforementioned variables as `constant`.

Remediation (revised commit: 5f11f2) : The aforementioned variables are now declared as `constant`.

F-2023-0171 - Unused State Variables - Info

Description: It was identified that solution has single state variable declared, but never used: `baseSpeed`. Such implementation increases Gas consumption during deployment and it occupies additional storage slot.

```
uint baseSpeed = 100000;
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Recommendations

Recommendation: It is recommended to remove unused state variable.

Remediation (revised commit: 5f11f2) : The unused state variable is now removed.

F-2023-0172 - The getLockTime() is redundant getter - Info

Description: The `getLockTime()` function appears to be redundant getter for the `locktime` state variable. Setting accurate state variable access makes getters redundant. Current implementation leads to extensive Gas usage during both deployment and usage.

```
function getLockTime() public view returns (uint256) {
    return locktime;
}
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Accepted

Recommendations

Recommendation: It is recommended to remove redundant getter and use global variable instead.

Remediation (revised commit: n/a) : The Client's team acknowledged this finding.

F-2023-0174 - Gas over-consumption in loops - Info

Description: Within the `sumRewardDebt()` function In the loop the counter variable is incremented using *i++* instead of *++i*. It is a known fact that using *++i* within `unchecked` clause costs less Gas per iteration inside the loops.

```
function sumRewardDebt() public  returns (uint256) {
[...]
        if(totalCumulative == 0){
           for (uint256 i = 0; i < addressList.length; i++) {
[...]
             }
        }
[...]
    }
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Recommendations

Recommendation: It is recommended to use *++i* with `unchecked` clause instead of *i++* inside the loops.

Remediation (revised commit: 5f11f2) : The *++i* operator is now used inside the loops.

F-2023-0175 - Redundant input parameter within the calculateDay() function - Info

Description: The `secondsPerDay` input variable within the `calculateDay()` function can be considered redundant, as second per day is a constant value of 86400. Thus, such implementation leads to extensive Gas consumption.

```
function calculateDay(
        uint256 timestamp,
        uint256 startTimestamp,
        uint256 secondsPerDay
    ) internal pure returns (uint256) {
        // require(timestamp >= startTimestamp, "calculateDay failed.");
        uint256 day = (timestamp - startTimestamp) / secondsPerDay;
        if ((timestamp - startTimestamp) % secondsPerDay > 0) {
            day += 1;
        }
        return day;
    }
```

Assets:

- mineWOW.sol [https://github.com/ullaniubility/ulla-contract]

Status: Fixed

Recommendations

Recommendation: It is recommended to remove redundant input variable and use `days` time unit instead.

Remediation (revised commit: 5f11f2) : The redundant input variable is now removed.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with 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).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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 recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

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

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/ullaniubility/ulla-contract
Commit	ab46260719de5ca70cee6dad21336bc81b274ae9
Whitepaper	Not provided.
Requirements	Not provided.
Technical Requirements	Not provided.

Contracts in Scope

./src/mineWOW.sol

The scope provided for remediation phase is as follow:

Scope Details

Repository	https://github.com/ullaniubility/ulla-contract
Commit	948570380834c8450924584aba6998874a17d0cc
Whitepaper	Not provided.
Requirements	Not provided.
Technical Requirements	Not provided.

Contracts in Scope

./src/mineWOW.sol