QuillAudits

Audit Report
February, 2021

OpenDeFi
by OroPocket

Contents

Scope of Audit

The scope of this audit was to analyze and document Unifarm smart contract codebase for quality, security, and correctness.

Check Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

Every issue in this report has been assigned with a severity level. There are four levels of severity and each of them has been explained below.

High severity issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality and we recommend these issues to be fixed before moving to a live environment.

Medium level severity issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems and they should still be fixed.

Low level severity issues

Low level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity four issues which indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	1	0
Closed	3	5	7	0

Introduction

On **February 18th, 2021** - Quillhash Team performed a final security audit for **Unifarm** smart contracts.
The code for the audit was taken from following the official GitHub link:
https://github.com/themohitmadan/unifarmv1_contract

Issues Found – Code Review / Manual Testing

A. Unifarm.sol

About the Contract:
Unifarm.sol is primarily a Staking contract that allows users to stake as well as claim their tokens after a particular period along with some rewards.

High severity issues

1. Stake Limit logic is Broken. Users can stake more than allowed

Line no - 73
Status: CLOSED

Description:
The stake function, includes a wrong require statement to check the stake limit of a particular user.

```
73          require(
74              userTotalStaking[msg.sender][tokenAddress] <=
75                  tokenDetails[tokenAddress].userStakeLimit,
76              "STAKE : Amount should be within permit"
77          );
```

This require statement simply compares the userTotalStaking mapping with the userStakeLimit of a user without taking into consideration the amount parameter passed by the user, i.e., the actual amount of tokens, the user is about to stake.

This will result in an unintended behaviour which lets the users stake more tokens than they are actually allowed to.

Recommendation:
In order to correctly check the stake limit condition, the require statement should be updated as follows:

```
73          require(
74              userTotalStaking[msg.sender][tokenAddress].add(amount) <=
75                  tokenDetails[tokenAddress].userStakeLimit,
76              "STAKE : Amount should be within permit"
77          );
```

2. **Wrong parameter passed to “getOneDayReward” function**
   Line no - 167
   Status: CLOSED

   Description:
   The getOneDayReward function expects 3 arguments, i.e.,
   - the amount of tokens staked
   - the address of the staked token
   - the address of reward token.

   However, while calling this function from the **_rewardCalculation** function, the address of the staked token itself is passed instead of the address reward token.

```
167                 getOneDayReward(
168                     stakedAmount,
169                     stakingDetails[msg.sender].tokenAddress[stakeId],
170                     stakingDetails[msg.sender].tokenAddress[stakeId]
171                 )
172             );
173
```

This will lead to unexpected behaviour and return a wrong value of reward since the **tokenDailyDistribution** mapping is designed to hold the distribution amount for a s**takedToken-rewardToken pair** rather than **stakedToken-stakedToken pair.**

Recommendation:
The right parameters should be passed while calling the getOneDayReward function in order to avoid the wrong calculation of the reward amount.

3. Reward amount is sent Twice within the rewardCalculationfunction

Line no - 173

Status: CLOSED

Description:

The _rewardCalculation function sends the reward to the referral address as well as the user address twice within the same function.

The function uses an ERC20 transfer to send the referral rewards at **Line 180 as well as Line 223**. Moreover, it sends the rewards to the user, using the sendToken function, at **Line 187 and Line 231.**

Is this behaviour intended?

Recommendation:

The procedure of calculating and sending the rewards must be checked.

If the above-mentioned process is not intended behaviour, the function must be updated with the right logic.

Medium severity issues

4. Events Emitted After External Call

Line no - 121,331,281

Status: CLOSED

Description:

As per the Check Effects Interaction Pattern, events should always be emitted before the external call.

The following functions emit events after the external call:

- **stake** function at line **121**
- **unStake** function at line **331**
- **sendToken** function at line **281**

Recommendation:

The Check Effects Interaction Pattern must be followed while including external calls in the function.

5. **Return Value of External call is never used**

Line no - 118, 180, 223, 278

Status: CLOSED

Description:

The return value of the externals made in the following functions are never used:

- **stake** function at line **118**
- **_rewardCalculation** function at line **180**
- **_rewardCalculation** function at line **223**
- **sendToken function** at line **278**

Recommendation:

Effective use of all return values must be ensured within the contract.

Low level severity issues

6. **External visibility should be preferred**

Line no - 344, 373

Status: CLOSED

Description:

Those functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.

This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as **external** within the contract:

- viewStakingDetails
- viewAvailableRewards

Recommendation:

The above-mentioned functions should be assigned external visibility.

7. **Comparison to boolean Constant**

   Line no - 168-170 , 245, 273, 302

   Status: OPEN

   Description:

   Boolean constants can directly be used in the conditional statements. Therefore, it's not considered a better practise to explicitly use **TRUE or FALSE** for comparisons.

   Recommendation:

   The equality to boolean constants must be eradicated from the above-mentioned line.

8. **Compiler version should be fixed**

   Solidity source files indicate the versions of the compiler they can be compiled with. It’s recommended to lock the compiler version in code, as future compiler versions may handle certain language constructions in a way the developer did not foresee. Also, it's recommended to use the latest compiler version.

B. Admin.sol

About the Contract:
Admin.sol smart contract includes all the functionalities for the owner of the contracts to maintain as well events emitted as modifying the imperative details of a token. It allows the admin to whitelist tokens, set token reward sequence, add new tokens etc.

Medium severity issues

1. **Redundant Variable Update**
   Line no - 159
   Status: CLOSED

   Description:
   The locableToken function includes redundant updation of a variable at Line 159.

   The **lockableDays** is, by default, **ZERO** at the time of initializing the **tokenInfo** struct.

```
158             );
159         } else if (lockableStatus == 2)
160             tokenDetails[tokenAddress].lockableDays = 0;
```

Therefore, it's not a better practice to initiate the **lockableDays** with **zero** once again.
Moreover, this unnecessary line of code consumes extra gas thus affecting the gas optimization part of the function.

Recommendation:
Eradicate redundant procedures in a function.

2. **No Events emitted after imperative State Variable modification**
   Line no - 164, 168,172
   Status: CLOSED

Description:
Functions that update an imperative arithmetic state variable contract should emit an event after the updation.
The following functions modify some crucial arithmetic parameters **like stakeDuration, refPercentage, optionableBenefit** etc in the Admin contract but don't emit any event after that.

Since there is no event emitted on updating these variables, it might be difficult to track it off-chain.

Recommendation:
An event should be fired after changing crucial arithmetic state variables.

3. **Return Value of External call is never used**
   Line no - 213
   Status: CLOSED

   Description:
   The return value of the externals made in the following functions are never used:
   - **safeWithdraw** function at Line **213**

   Recommendation:
   Effective use of all return values must be ensured within the contract.

Low level severity issues

4. **External visibility should be preferred**
   Status: CLOSED

   Description:
   Functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.

   This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as external within the contract:

- addToken
- setDailyDistribution
- updateSequence
- updateToken
- updateWhiteListStatus
- lockableToken
- updateStakeDuration
- updateOptionableBenefit
- updateRefPercentage
- updateIntervalDays
- changeTokenBlockedStatus
- safeWithdraw
- renounceOwnership()
- transferOwnership

Recommendation:
The above-mentioned functions should be assigned external visibility.

5. **Comparison to boolean Constant**
   Line no - 49, 79-83, 95-98, 104-107, 118, 152, 191-195
   Status: CLOSED

   Description:
   Boolean constants can directly be used in the conditional statements.

   Therefore, it's not considered a better practise to explicitly use TRUE or FALSE for comparisons.

   Recommendation:
   The equality to boolean constants must be eradicated from the above-mentioned line.

6. Compiler version should be fixed

Satus: CLOSED

Solidity source files indicate the versions of the compiler they can be compiled with. It’s recommended to lock the compiler version in code, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

C. UnifarmFactory.sol

About the Contract:
UnifarmFactory is basically a Factory Contract that allows, only the owner, to deploy new instances of the Unifarm smart contracts. It stores all the addresses of the new contracts created in an array.

High severity issues

None Found

Medium severity issues

None Found

Low level severity issues

1. External visibility should be preferred
   Status: CLOSED

   Description:
   Functions that are never called throughout the contract should be marked as external visibility instead of public visibility.

   This will effectively result in Gas Optimization as well.

   Therefore, the following function must be marked as external within the contract:
   - deploy

2. **Compiler version should be fixed**
   Status: CLOSED

   Solidity source files indicate the versions of the compiler they can be compiled with. It's recommended to lock the compiler version in code, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.
During the process of audit, several issues of high, medium as well as low severity were found which might affect the intended behaviour of the contracts.
However, the majority of the issues have now been fixed.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Unifarm platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Unifarm Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

OpenDeFi

by OroPocket

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com