



QuillAudits



Audit Report April, 2021

OpenDeFi
by  OroPocket

Contents

Introduction	01
Techniques and Methods	02
Issue Categories	04
Issues Found – Code Review/Manual Testing	06
Automated Testing	08
Closing Summary	09
Disclaimer	10

Overview

Unifarm Token

- ERC-20 Token
- Token Name: UNIFARM Token
- Token Symbol: UFARM
- Total Supply of 1,000,000,000 UNIFARM (1 billion Unifarm tokens). Mint function should only be used once, after that mint function should not exist for the admin.
- Admin should be able to lock token for particular addresses.
- Admin should be able to burn token at particular address.
- In built governance model where 1 UNIFARM = 1 Vote

UnifarmToken.sol:

<https://rinkeby.etherscan.io/address/0x2eb2d3a3cab6bb3ced92625e4a8cf054748d055b#code>

Scope of Audit

The scope of this audit was to analyse UnifarmToken.sol smart contract's codebase for quality, security, and correctness.

Check Vulnerabilities

We have scanned the Unifarm Token smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Signature Malleability
- EIP72 Structures
- Replay Attacks
- Use of ecrecover() function
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply

- Integer overflow/underflow
- ERC20 transfer() does not return
- boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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 intended behaviour mentioned in whitepaper.
- 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.

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

Mythril, Slither, SmartCheck, Solhint.



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	3	3
Closed	0	1	2	6

Issues Found – Code Review / Manual Testing

Best practices

ecrecover does not check for malleable signatures. `s` values that are part of accepted signatures should be checked to be in lower ranges. The recommended procedure for using the **ecrecover** function can be found in Open Zeppelin's excellent ECDSA library.

Ref:

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/cryptography/ECDSA.sol>

[FIXED][#L610-612] function **delegate()** can **return true** in context to handling operations

High severity issues

None.

Medium severity issues

- **[FIXED]**Access Control Mechanism Flaw

Considering Admin and Owner to be two different addresses. The Admin can take over the owner permissions by calling function **renounceOwnership()**, now the Previous Owner has left with no powers or in other words not a Powerful Role Anymore, because it can not even call the **transferOwnership()** function, as it is not an Owner Anymore. Now the Admin is the Most Powerful Role, by having both the Owner and Admin Role

Considering Admin and Owner to be the same addresses. Once the Ownership is transferred by calling **transferOwnership()** to a new Owner from the current owner. The previous owner will still be the Admin, and can call **renounceOwnership()** to take control of the ownership from the new owner

Missing: Proper Access Control Mechanisms

Status: Fixed

Reason: Intended logic as provide by Developer

Comments from Developer: Admin and Owner both are the same as we need to have dual ownership functionality to prevent any situation where we lose the ownership or in bad hands. then admin can call renounce ownership to take ownership back.

Low level severity issues

- [#6] older compiler used. Use newer versions to avoid bugs introduced in older compilers
- [#556, 630] **block.timestamp** used for comparisons. Avoid using it as it can be manipulated by miners
- **[FIXED]**[705] wrong check
require(balances[holder] >= amount, "Insufficient balance");
it should be
require(lockedTokens[holder] >= amount, "Insufficient Tokens To Unlock");

Can cause revert on Subtraction Overflow, if the supplied amount is more than the number of locked tokens of the holder

- **[FIXED]**[#L590-604] function **transferFrom()**, Zero Address Checks can be added to avoid revert on subtraction overflow at **#L596**: Add checks for **spenderAllowance**, such that it should be greater than **rawAmount**
- **ERC20 approve() race**
The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval, and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.

Possible Solution: make sure to create user interfaces in such a way that they set the allowance first to 0 before setting it to another value for the same spender

Reference:

- https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit
- <https://medium.com/mycrypto/bad-actors-abusing-erc20-approval-to-steal-your-tokens-c0407b7f7c7c>
- <https://eips.ethereum.org/EIPS/eip-20>

Informational

- **[FIXED]**[#384] use `_msg.sender()`; function from context library instead of using `msg.sender`; directly
- **[FIXED]**[#401] it should be "Caller is not the **Admin**"
- [#406-411] Edit developer comments as well, after making changes in the corresponding functions. The comment doesn't make any sense and not matches with the corresponding function
- **[FIXED]**[#412]**renounceOwnership** should be called by **onlyOwner**
There can be separate functions for transferring Admin role to the new Admin, and also renouncing the current admin role

Reason: Intended logic as provide by Developer

- **[FIXED]**[#L518] zero address checks can be added for account and spender in function **approve()**
- [#671] **while (upper > lower)** . Unknown upper bound **uper** can result in extra gas in loops

- **[FIXED]**[#713] zero address checks can be added for delegatee address in function `_delegate()`
- **[FIXED]**[#723] function `_transferTokens()`: checks can be added for `amount > 0`
- [#755] Strict Equality has been used

Gas Optimization

Public functions that are never called by the contract should be declared external to save gas.

- **[FIXED]**[#L610-612] function `delegate()` and [#L623-632]function `delegateBySig()`: Visibility can be limited to **external**

Automated Testing

Slither

Slither didn't detect any issues

Mythril

Mythril didn't detect any issues

Smartcheck

SmartCheck didn't detect any issues

Solhint

```
UnifarmToken/UnifarmToken.sol
354:2  error  Line length must be no more than 120 but current length is 132  max-line-length
468:2  error  Line length must be no more than 120 but current length is 127  max-line-length
471:2  error  Line length must be no more than 120 but current length is 122  max-line-length
474:2  error  Line length must be no more than 120 but current length is 142  max-line-length
542:2  error  Line length must be no more than 120 but current length is 124  max-line-length
550:2  error  Line length must be no more than 120 but current length is 126  max-line-length
551:2  error  Line length must be no more than 120 but current length is 122  max-line-length
624:2  error  Line length must be no more than 120 but current length is 126  max-line-length
```

```
* 8 problems (8 errors, 0 warnings)
```


Closing Summary

Several issues of medium and low severity have been reported during the audit, out of which, most of them have been fixed. Some suggestions have also been made to improve the code quality and gas optimization. There were NO critical or major issues found that can break the intended behavior. Given the subjective nature of some assessments, it will be up to the Unifarm team to decide whether any changes should be made.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code. Besides, a security audit, please don't consider this report as investment advice.

OpenDeFi
by OroPocket



QuillAudits

📍 Canada, India, Singapore and United Kingdom

💻 audits.quillhash.com

✉️ hello@quillhash.com

