QuillAudits

Audit Report
May, 2021

UniFarm
by OroPocket

Contents

Introduction

During the period of **May 18th, 2021 to May 21th, 2021** – QuillHash Team performed security audits for **Unifarm Vesting** smart contracts. The code for audit was taken from the following link:

https://github.com/themohitmadan/unifarm_vesting/blob/master/contracts/UFARMBeneficiaryBook.sol
[Commit Hash: 629194ca1a7f9fb3bfc7e6f28a65446c5f5c7eb7]
Updated : 64f6f033b61d998d45ef5ff61e6db585ce1790c0

https://github.com/themohitmadan/unifarm_vesting/blob/master/contracts/UnifarmFactory.sol
[Commit Hash: 629194ca1a7f9fb3bfc7e6f28a65446c5f5c7eb7]
Updated : 64f6f033b61d998d45ef5ff61e6db585ce1790c0

https://github.com/themohitmadan/unifarm_vesting/blob/master/contracts/UnifarmVesting.sol
[Commit Hash: 629194ca1a7f9fb3bfc7e6f28a65446c5f5c7eb7]
updated : 64f6f033b61d998d45ef5ff61e6db585ce1790c0

Overview of Unifarm Vesting

UniFarm is a decentralized farming pool of DEFI's top projects. Common staking/farming programs allow users to farm only the token that they stake. Here they can farm multiple tokens. Users can unstake anytime, there is no Lock-in. It provides one Place to Farm Instead of having to go to different exchanges etc, to stake their different holdings, users can easily farm from one place while earning multiple. Tokens are held in a self-custodian Wallet like Metamask—decentralised.

The features & utility of the UniFarm ecosystem

- Simple UI. One place to farm all.
- No market exposure of staked tokens.
- Stake one, farm many tokens.
- Working jointly with several projects gives a wider audience to all
- participating projects in the pool.
- Create multiple staking pools.
- Decentralized.
- Gamification done in a way to reduce any sell pressure.

Unifarm Token

- Governance token for all protocol level changes.
- UniFarm tool will be added to all premium pools enabling token holders
- to stake UNIFARM and earn a bunch of rewards.
- UniFarm is already cash flow positive.
- 25% of the revenue will be used to buy back UNIFARM tokens and lock
- them for a year.
- Total supply of 1 Billion tokens.
- 50% of the supply is used for farming.

Tokenomics

Token Ticker: $UFARM

Total Supply: 1,000,000,000 $UFARM

Cap Table

Item	Number of tokens	Percent of tokens	Unlock Schedule
Fundraise	140,000,000	14%	Variable for different rounds. Details below
Liquidity for exchanges	50,000,000	5%	Unlocked for Balancer/Exchange liquidity/IDO. Liquidity will be added at $0.04
Farming	500,000,000	50%	UniFarm pool farming
Team	100,000,000	10%	Cliff for 1 year, monthly unlock from 13-24 months
Advisors	50,000,000	5%	Cliff for 6 months, daily linear unlocking from days 181-360
Marketing	50,000,000	5%	Cliff for 3 months; Monthly unlocking from month 4-24
Company Reserve	60,000,000	6%	Cliff for 6 months; Quarterly unlock over 6 quarters
Community	50,000,000	5%	Variable for community. Details below

Fundraise

Token sale of UniFarm

Round	Price per token	Total token allocation	Total Raise	Unlock schedule
Seed	$0.01	50,000,000	$500,000	5% on TGE; Linear daily unlock from day 90 to 360.
Private	$0.02	90,000,000	$1,800,000	10% on TGE; Linear daily unlock from day 90 to 360.

Community

Token allocation for community

Type	Number of tokens	Distribution	Unlock Schedule
ORO Token Holders	25,000,000	ORO Token holders will be able to claim $UFARM tokens. This will be done quarterly. More details would be announced.	Quarterly airdrops.
UniFarm Pool Farmers	12,500,000	$UFARM tokens fair launched to all that have participated in UniFarm farms (max 2000 wallets). More details to be announced soon.	20% quarterly.
UNIFARM Pool for ORO	12,500,000	High APY pool for ORO token holders to stake ORO tokens and farm UniFarm AND High APY pool for Liquidity providers on AMMs. More details will be announced soon.	Variable

Scope of Audit

The scope of this audit was to analyse the Unifarm Vesting smart contract codebase for quality, security, and correctness. Following is the list of smart contracts included in the scope of this audit:

- UFARMBeneficiaryBook.sol
- UnifarmFactory.sol
- UnifarmVesting.sol

OUT-OF-SCOPE: External contracts, External Oracles, other smart contracts in the repository or imported smart contracts, economic attacks.

Checked Vulnerabilities

We have scanned Unifarm Vesting smart contracts 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
- DoS with (Unexpected) Throw
- DoS with (Unexpected) revert
- 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
- Address hardcoded
- Using delete for arrays
- Integer overflow/
- underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of Unifarm Vesting 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 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

A combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

Structural Analysis
In this step, we have analysed 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.

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 analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

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

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

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

Assessment Summary and Findings Overview

High severity issues

Issues that must be fixed before deployment else they can create major issues.

Medium level severity issues

These issues will not create major issues in working but affect the performance of the smart contract.

Low level severity issues

These issues are more suggestions that should be implemented to refine the code in terms of gas, fees, speed and code accuracy

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	0	3
Closed	0	0	7	1

Findings and Tech Details

A. UFARMBeneficiaryBook Contract

High severity issues

No Issue found under this category

Medium severity issues

No Issue found under this category

Low level severity issues

1. **The compiler version should be fixed**

   Code Lines: 3
   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. It is recommended to use fixed compiler version 0.7.6.

   Status: Fixed

2. **Consider using the latest stable solidity version**

   Code Lines: 3
   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Status: Fixed

3. **Use local variable instead of state variable like length in a loop**

   Code Lines: 254
   For every iteration of for loop state variables like the length of the non-memory array will consume extra gas. To reduce gas consumption, it's advisable to use local variables.

   Status: Fixed

B. UnifarmFactory Contract

High severity issues

No Issue found under this category

Medium severity issues

No Issue found under this category

Low level severity issues

1. **The compiler version should be fixed**

   Code Lines: 3
   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. It is recommended to use fixed compiler version 0.7.6.

   Status: Fixed

2. **Consider using the latest stable solidity version**

   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Status: Fixed

Informational

1. **Coding Style Issues**

   Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It’s recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

```
UnifarmFactory.sol
  38:2  error  Line length must be no more than 120 but current length is 134  max-line-length

✖ 1 problem (1 error, 0 warnings)
```

Status: Fixed

C. UnifarmVesting Contract

High severity issues

No Issue found under this category

Medium severity issues

No Issue found under this category

Low level severity issues

1. **The compiler version should be fixed**

   Code Lines: 3
   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. It is recommended to use fixed compiler version 0.7.6.

   Status: Fixed

2. **Consider using the latest stable solidity version**

   The smart contract is using an old version of solidity. It is highly recommended to use the latest stable version of solidity.

   Status: Fixed

Informational

1. **Coding Style Issues**

   Coding style issues influence code readability and, in some cases, may lead to bugs in future. Smart Contracts have a naming convention, indentation and code layout issues. It's recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

```
UnifarmVesting.sol
   95:2  error  Line length must be no more than 120 but current length is 157  max-line-length
  127:2  error  Line length must be no more than 120 but current length is 157  max-line-length
  128:2  error  Line length must be no more than 120 but current length is 180  max-line-length
  232:2  error  Line length must be no more than 120 but current length is 126  max-line-length

✖ 4 problems (4 errors, 0 warnings)
```

2. Consider simplifying the unlockedDays calculation

   Code Lines: 206, 207
   Consider using a variable to store the calculation instead of doing calculations in the ternary operator.

3. **Consider using require instead of revert inside if..else**

   Code Lines: 157-173
   In function distribute() consider using require to check the condition. The two ways if (!condition) revert(...); and require(condition, ...); are equivalent. Use of require provides more readability.

Automated Testing

Slither

Slither is an open-source Solidity static analysis framework. This tool provides rich information about Ethereum smart contracts and has the critical properties. It runs a suite of vulnerability detectors, prints visual information about contract details, and provides an API to easily write custom analyses. Slither enables developers to find vulnerabilities, enhance their code comprehension, and quickly prototype custom analyses.

Slither didn't raise any critical issue with smart contracts. The smart contracts were well tested and all the minor issues that were raised have been documented in the report. Also, all other vulnerabilities of importance have already been covered in the **Findings and Tech Details** section of the report.

UFARMBeneficiaryBook.so

```
INFO:Printers:
Compiled with solc
Number of lines: 319 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 4 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 4
Number of informational issues: 1
Number of low issues: 0
Number of medium issues: 0
Number of high issues: 0


+-------------------------+-------------+------+------------+--------------+----------+
|          Name           | # functions | ERCS | ERC20 info | Complex code | Features |
+-------------------------+-------------+------+------------+--------------+----------+
| UFARMBeneficiaryBook    |      17     |      |            |      No      |          |
+-------------------------+-------------+------+------------+--------------+----------+
```

UnifarmFactory.sol

```
INFO:Printers:
Compiled with solc
Number of lines: 1175 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 10 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 4
Number of informational issues: 9
Number of low issues: 6
Number of medium issues: 2
Number of high issues: 0

ERCs: ERC20
```

Name	# functions	ERCS	ERC20 info	Complex code	Features
UnifarmFactory	12			No	
UnifarmVesting	20			No	Send ETH Tokens interaction
IERC20	6	ERC20	No Minting Approve Race Cond.	No	
IUFARMBeneficiaryBook	1			No	
Address	11			No	Send ETH Delegatecall Assembly
SafeERC20	6			No	Send ETH Tokens interaction
SafeMath	13			No	

UnifarmToken.sol

```
INFO:Printers:
Compiled with solc
Number of lines: 1010 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 6 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 2
Number of informational issues: 3
Number of low issues: 3
Number of medium issues: 1
Number of high issues: 0

ERCs: ERC20
```

Name	# functions	ERCS	ERC20 info	Complex code	Features
UnifarmToken	33	ERC20	No Minting Approve Race Cond.	Yes	Ecrecover Assembly
IERC20	6	ERC20	No Minting Approve Race Cond.	No	
SafeMath	13			No	

UnifarmVesting.sol

```
INFO:Printers:
Compiled with solc
Number of lines: 1096 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 9 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 3
Number of informational issues: 9
Number of low issues: 6
Number of medium issues: 2
Number of high issues: 0

ERCs: ERC20
```

Name	# functions	ERCS	ERC20 info	Complex code	Features
UnifarmVesting	20			No	Send ETH Tokens interaction
IERC20	6	ERC20	No Minting Approve Race Cond.	No	
IUFARMBeneficiaryBook	1			No	
Address	11			No	Send ETH Delegatecall Assembly
SafeERC20	6			No	Send ETH Tokens interaction
SafeMath	13			No	

Mythril

Mythril is a security analysis tool for EVM bytecode. It detects security vulnerabilities in smart contracts built for Ethereum. It uses symbolic execution, SMT solving and taint analysis to detect a variety of security vulnerabilities.
Mythril did not detect any high severity issue. All the considerable issues raised by Mythril are already covered in the **Findings and Tech Details** section of this report

Manticore

Manticore is a symbolic execution tool for the analysis of smart contracts and binaries. During Symbolic Execution / EVM bytecode security assessment did not detect any high severity issue. All the considerable issues are already covered in the **Findings and Tech Details** of this report.

Closing Summary

Overall, the smart contract code is extremely well documented, follows a high-quality software development standard, contains many utilities and automation scripts to support continuous deployment/ testing/ integration, and does NOT contain any obvious exploitation vectors that QuillHash was able to leverage within the timeframe of testing allotted. Overall, the smart contracts adhered to **ERC20** guidelines. No critical or major vulnerabilities were found in the audit. **Several issues of low severity were found and reported during the audit, most of which are resolved now.**

The outcome of this security audit is satisfactory; due to time and resource constraints, only testing and verification of essential properties were performed to achieve objectives and deliverables set in the scope. QuillHash recommends performing further testing to validate extended safety and correctness in context to the whole set of contracts.

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.

UniFarm

by OroPocket

QuillAudits

Canada, India, Singapore and Uniked Kingdom

audits.quillhash.com

audits@quillhash.com