SMART CONTRACT AUDIT

ZOKYO.

June 8th 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

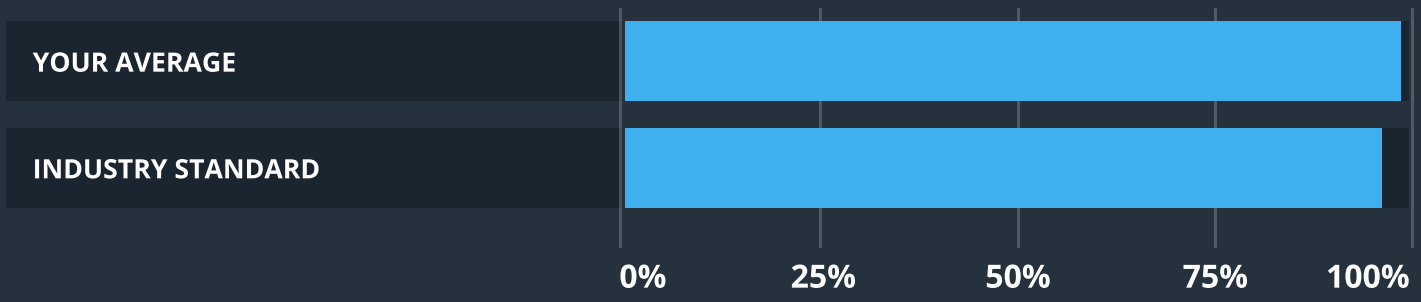
TECHNICAL SUMMARY

This document outlines the overall security of the UniFarm smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

Testable Code



The testable code is 98%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the UniFarm team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the UniFarm repository.

Repository - https://github.com/themohitmadan/unifarmYF2

Last commit - e847a08716a2abf89893e6d5cff5d114b6bf2055

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- UnifarmCohort.sol
- UnifarmNFTManagerUpgradeable.sol
- Multicall2.sol
- UnifarmCohortFactoryUpgradeable.sol
- UnifarmRewardRegistryUpgradeableStorage.sol
- UnifarmRewardRegistryUpgradeable.sol
- UnifarmCohortRegistryUpgradeable.sol
- UnifarmNFTDescriptorUpgradeable.sol
- UnifarmCohortRegistryUpgradeableStorage.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of UniFarm smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

There were no critical issues found during the audit. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

Contracts are well written and structured. The findings during the audit have no impact on contract performance or security, so it is fully production-ready.

Despite the fact, the expected logic is managing all vestings by the owner, it should be careful with parameters to avoid mistakes during the vesting process.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

MEDIUM | RESOLVED

In contract UnifarmNFTManagerUpgradeable, function stakeOnUnifarm, variable farmToken it's an user input that represents an address that it is not sanitized before, this lead to re-entracy attacks and unpredictable behaviours.

Recommendation:
Whitelist the farmToken address or ad re-entracy attacks to the stake and burn function and the other functions that modified the same state data that the stake function does. (Done)

LOW | RESOLVED

Contracts UnifarmCohort, UnifarmNftManagerUpgradeable, etc... are making use of SafeMath for mathematical operations, but since the solidity 0.8.0, the use of SafeMath is not necessary anymore because the revert happens automatically in case of overflow or underflow, so SafeMath is not adding any benefits and it's just making taking more gas with no addeds benefits.

Recommendation:
Remove the use of SafeMath library and the use of it's methods and replace them with normal signs calculations. (Done we removed the SafeMath Now)

INFORMATIONAL | UNRESOLVED

In contract UnifarmCohort, functions setPortionAmount and disableBooster does not contains sanity checks, even if they are only settable by the owner they still need to contains sanity checks to be in standard with best practices.

Recommendation:
Add sanity checks to the setPortionAmount and disableBooster functions. (this was informational so skipping it)

INFORMATIONAL | RESOLVED

In contract UnifarmCohort, function safeWithdrawAll, for a better gas optimization, you can safe the value of tokens.length in memory and iterate over the variable that is safe in memory, it will be cheaper to read from memory every time then to read from storage.

Recommendation:
Create a new variable that it will store the value of tokens.length and use it in the while condition. (Done)

INFORMATIONAL | RESOLVED

In contract UnifarmRewardRegistryUpgradeable, function safeWithdrawAll, for a better gas optimization, you can safe the value of tokens.length in memory and iterate over the variable that is safe in memory, it will be cheaper to read from memory every time then to read from storage.

Recommendation:
Create a new variable that it will store the value of tokens.length and use it in the while condition. (Done)

INFORMATIONAL | RESOLVED

In contract UnifarmRewardRegistryUpgradeable, function setRewardCap, for a better gas optimization, you can save the value of rewardTokenAddresses.length in memory and iterate over the variable that is safe in memory, it will be cheaper to read from memory every time then to read from storage.

Recommendation:
Create a new variable that it will store the value of rewardTokenAddresses.length and use it in the for loop. (Done)

INFORMATIONAL | RESOLVED

In contract UnifarmRewardRegistryUpgradeable, function sendMulti, for a better gas optimization, you can save the value of rewardTokens.length in memory and iterate over the variable that is safe in memory, it will be cheaper to read from memory every time then to read from storage.

Recommendation:
Create a new variable that it will store the value of rewardTokens.length and use it in the for loop. (Done)

INFORMATIONAL | RESOLVED

In contract UnifarmRewardRegistryUpgradeable, function addInfluencers, for a better gas optimization, you can safe the value of userAddresses.length in memory and iterate over the variable that is safe in memory, it will be cheaper to read from memory every time then to read from storage.

Recommendation:
Create a new variable that it will store the value of userAddresses.length and use it in the while condition. (Done)

INFORMATIONAL | RESOLVED

In library ConvertHexToString, function toString, for a better gas optimization, you can save the value of data.length in memory and iterate over the variable that is safe in memory, it will be cheaper to read from memory every time then to read from storage.

Recommendation:
Create a new variable that it will store the value of data.length and use it in the for loop. (Done)

INFORMATIONAL | RESOLVED

Contract UnifarmCohort, variable factory is assigned only in the constructor and then the value is used all over the contract, from this behavior we understand that the scope of the factory variable is that it should be assigned only in contract constructor, for that we would recommend making the variable as 'immutable', (immutable variables can only be assigned inside the constructor and only there), for extra security and to be in accordance with best practices.

Recommendation:
Make variable factory from line 47 as immutable. (Done)

INFORMATIONAL | UNRESOLVED

In contract UnifarmCohort, all the revert messages form the require functions are very short and hard to understand, we understand the reason for this was the gas optimization but this is not necessary, the strings in solidity are defined on 32 bytes, so even if your string message hase only 2 characters in it, it will still take 32 bytes, so our recommendation would be to make the messages more clear and to take in consideration to not make them any longer then 32 bytes, if you can make them explicit and with a length of 32 bytes it will take as many gas as this short messages are taking right now.

Recommendation:
Make the messages more explicative but no longer then 32 bytes because they are already defined on 32 bytes, so there is no added benefit to make them shorter then 32 bytes, prioritize clarity and explanations. (this was informational so skipping it)

INFORMATIONAL | UNRESOLVED

In contract UnifarmNftManagerUpgradeable, all the revert messages form the require functions are very short and hard to understand, we understand the reason for this was the gas optimization but this is not necessary, the strings in solidity are defined on 32 bytes, so even if your string message hase only 2 characters in it, it will still take 32 bytes, so our recommendation would be to make the messages more clear and to take in consideration to not make them any longer then 32 bytes, if you can make them explicit and with a length of 32 bytes it will take as many gas as this short messages are taking right now.

Recommendation:
Make the messages more explicative but no longer then 32 bytes because they are already defined on 32 bytes, so there is no added benefit to make them shorter then 32 bytes, prioritize clarity and explanations. (this was informational so skipping it)

	UnifarmCohortFactory Upgradeable.sol	UnifarmCohortRegistry Upgradeable.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	UnifarmNFTDescriptor Upgradeable.sol	UnifarmCohort.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	UnifarmNFTManager Upgradeable.sol	UnifarmRewardRegistry Upgradeable.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	Multicall2.sol	UnifarmRewardRegistry UpgradeableStorage.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	UnifarmCohortRegistryUpgradeableStorage.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

As part of our work assisting UniFarm in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the UniFarm contract requirements for details about issuance amounts and how the system handles these.

Multicall2
  Initialization
    ✓ Tests initialization
  Agreggate
    ✓ aggregates calls (158ms)
    ✓ try aggregates calls (129ms)
    ✓ try block and aggregates calls (124ms)
    ✓ try block and aggregates calls (121ms)
    ✓ getsBlockHash
    ✓ getBlockNumber
    ✓ getCurrentBlockCoinbase
    ✓ getCurrentBlockDifficulty
    ✓ getCurrentBlockGasLimit
    ✓ getCurrentBlockTimestamp
    ✓ getEthBalance
    ✓ getLastBlockHash
  blockAndAgg
    ✓ block And Aggregate
  tryBlockAndAgg
    ✓ tryBlockAndAggregate (57ms)
UnifarmNFTDescriptor
  Initialization
    ✓ Has correct initialization info
  generateTokenURI
    ✓ generateTokenURI (544ms)
NFTManager
  Initialization
    ✓ Tests initialization values (112ms)
  stakeOnUnifarm

✓ Should stake on unifarm (136ms)

unstake

✓ unstakeOnUnifarm (119ms)

SetFeeConfiguration

✓ updates feeConfigurations successfully (62ms)

tokenURI

✓ tokenURI (623ms)

claimOnUnifarm

✓ claimOnUnifarm (77ms)

emergencyBurn

✓ emergencyBurn (92ms)

buyBoosterPackOnUnifarm

✓ buyBoosterPackOnUnifarm (196ms)

stakeAndBuyBoosterPackOnUnifarm()

✓ stakes and buys booster pack on unifarm (146ms)

Registry

Initialization

✓ Should have correct master,forwarder, multical address

setTokenMetaData

✓ reverts with WFID, IFT, IC, ONA then sets token metadata and emit

TokenMetaDataDetails (148ms)

setCohortDetails()

✓ reverts with ICI, ONA, then it set Cohort Details (99ms)

addBoosterPackage()

✓ reverts with WBPID then Adds booster Package (83ms)

getBoosterPackDetails()

✓ gets booster pack details (83ms)

getCohortToken()

✓ gets correct cohortToken (81ms)

validateUnStake and unstake

✓ reverts with LC then it validates Unstake (87ms)

✓ reverts with LC then validate stake (73ms)

updateMulticall()

✓ reverts with SMA then updateMulticall

UnifarmRewardRegistryUpgradeable

Initialization

✓ should have correct initialization values (64ms)

update Ref Percentage

✓ reverts with IS then update Referal Percentage then emits UpdatedRefPercentage

updateMulticall()

✓ reverts with SMA then update multicall address

addInfluencers()

✓ adds list of Influencers and influencer percentages
setRewardCap()
✓ reverts with IS, ICI, IL, IRA then sets Reward Cap (57ms)
setRewardTokenDetails
✓ reverts with ICI then sets reward token details, emits SetRewardTokenDetails
getRewardTokens
✓ gets reward tokens and pbr (112ms)
getInfluencerReferralPercentage()
✓ gets correct influencer referral percentage
safeWithdrawAll()
✓ reverts with IWA, SF, and safe withdraws all (235ms)
safeWithdrawEth()
✓ safe Withdraws Eth (72ms)
distributeRewards
✓ distributeRewards (492ms)

UnifarmCohort

Initialization
✓ SUCCESS: should have correct factory adress
buyBooster()
✓ Should buy booster pack (145ms)
viewStakingDetails
✓ viewStakingDetails (59ms)
Permissions
✓ reverts with NOA
setPortionAmount
✓ sets portion amount (88ms)
safeWithdrawEth
✓ safeWithdrawEth (82ms)
safeWithdrawAll
✓ safeWithdrawAll successfully (280ms)
disableBooster
✓ disables booster (76ms)
collectPrematureRewards
✓ collectPrematureRewards (206ms)
stake()
✓ stakes (491ms)
unStake()
✓ unStake

UnifarmCohortFactory

Initialization
✓ SUCCESS: should have correct factory adress
transferOwnership

```
    ✓ Succesfully transfers ownership (140ms)
setStatesContracts
    ✓ Sets StatesContracts (44ms)
computesCohortAddress
    ✓ creates cohort succesfully (45ms)
createUnifarmCohort
    ✓ creates cohort succesfully (92ms)
get number of cohorts
    ✓ getsNumberOfCohort (69ms)
63 passing (38s)
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
UnifarmCohort.sol	93.81	76.09	100	94.12	133,134, 350
UnifarmCohortFactory Upgradeable.sol	100	100	100	100	
UnifarmCohortRegistry Upgradeable.sol	100	92.31	100	100	
UnifarmNFTDescriptor Upgradeable.sol	100	100	100	100	
UnifarmNFTManagerUp gradeable.sol	100	90	100	100	
UnifarmRewardRegistr yUpgradeable.sol	98.53	88.89	100	100	
Multicall2.sol	100	100	100	100	
UnifarmRewardRegistry UpgradeableStorage.sol	100	100	100	100	
UnifarmCohortRegistry UpgradeableStorage.sol	100	100	100	100	
All files	**97.71**	**86.49**	**100**	**98.1**	

We are grateful to have been given the opportunity to work with the UniFarm team.

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.

Zokyo's Security Team recommends that the UniFarm team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.