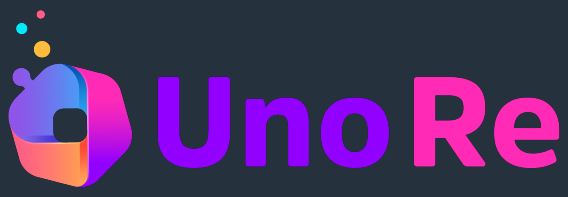


SMART CONTRACT AUDIT

ZOKYO.

September 3d, 2021 | v. 2.0

PASS

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

TECHNICAL SUMMARY

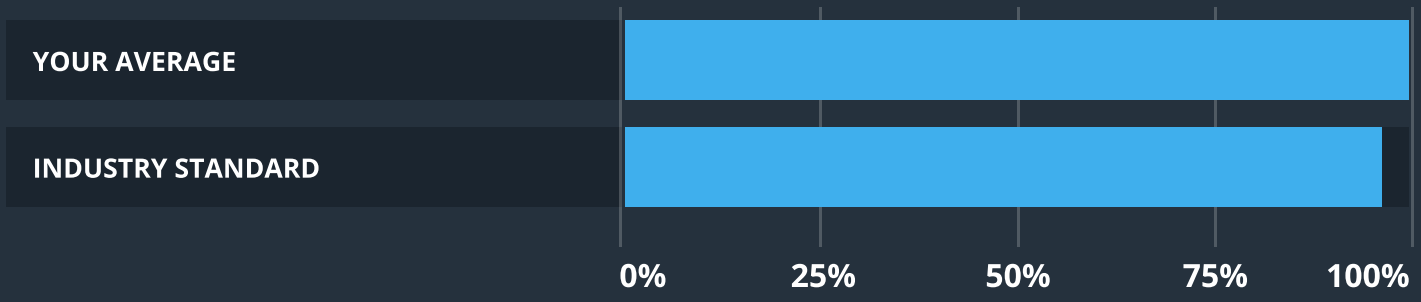
This document outlines the overall security of the Uno smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were no critical or high issues found during the audit.

Testable Code



The testable code is 100%, 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 security of the contract we at Zokyo recommend that the Uno 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 Uno smart contract's source code was taken from one archive provided by the Uno team

SHA-256 (audited): 2ae3fc057643ea29f52535a35f812d960b20780d5c52d615babd206e9dd530a2
SHA-256 (post-audit): 07192b84cea2dc6885804c2c2707427b3c6e47ba887a76c9b10b012059aae331

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

- Actuary.sol
- ClaimAssessor.sol
- CohortFactory.sol
- PremiumPoolFactory.sol
- PremiumPool.sol
- Cohort.sol
- UnoERC20.sol
- RiskPool.sol
- RiskPoolFactory.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;
- 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 Uno 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. 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 or high issues found during the audit. Although certain number of medium issues were discovered, they relate to the following:

- Return status ignored
- Incorrect event argument
- Use ReentrancyGuard for every external method.

After recommendations by Zokyo auditors, all issues that influence security and efficiency were fixed by Uno Team.

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. 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

Return status ignored

ClaimAssessor.requestClaim
At contracts/ClaimAssessor.sol#11-19
ignores return value by
ICohort(_cohort).requestClaim(_from,uint16(_protocolIdx),_amount)
contracts/ClaimAssessor.sol#18
If the return value is the success status, then it's important to check it.

Recommendation:
Check the return value.

MEDIUM | RESOLVED

Incorrect event argument

At contracts/Cohort.sol:228
The _amount is decreasing at lines
contracts/Cohort.sol:222
contracts/Cohort.sol:225
contracts/Cohort.sol:213
So the wrong value will be displayed in the event.

Recommendation:
Store total _amount in the other value and pass it to the event.

MEDIUM | RESOLVED

Use ReentrancyGuard for every external method

To completely prevent reentry attacks it's better to place a nonReentrant modifier for every state changing method.

Recommendation:
Go safe. Use nonReentrant for every external method.

LOW | RESOLVED

Function always returns the same value

At contracts/Cohort.sol:192
The function requestClaim always returns true.

Recommendation:
Remove the return statement because they make no difference.

LOW | RESOLVED

Multiply first, the divide

There is a multiplication on the result of a division:
-_pr = (((_totalPr * amount) / poolCapital[_pool]) * IRiskPool(_pool).APR()) / totalAPRofPools (contracts/Cohort.sol#180)
This operations order decreases the result accuracy.

Recommendation:
First multiply, then divide.

LOW | UNRESOLVED

Method should be declared external

If you don't use methods from inside the contract, then it's better to declare them external. It saves gas.
- Actuary.cohortCreatorsLength() (contracts/Actuary.sol#24-26)
- Cohort.allRiskPoolLength() (contracts/Cohort.sol#85-87)
- Cohort.changePoolPriority(uint8,uint8) (contracts/Cohort.sol#278-282)
- UnoERC20.decimals() (contracts/UnoERC20.sol#63-65)
- UnoERC20.transfer(address,uint256) (contracts/UnoERC20.sol#89-92)
- UnoERC20.allowance(address,address) (contracts/UnoERC20.sol#97-99)
- UnoERC20.approve(address,uint256) (contracts/UnoERC20.sol#108-111)
- UnoERC20.transferFrom(address,address,uint256) (contracts/UnoERC20.sol#126-140)
- UnoERC20.increaseAllowance(address,uint256) (contracts/UnoERC20.sol#154-157)
- UnoERC20.decreaseAllowance(address,uint256) (contracts/UnoERC20.sol#173-181)
- UnoERC20.decimals() (contracts/UnoERC20.sol#63-65)
- UnoERC20.totalSupply() (contracts/UnoERC20.sol#70-72)
- UnoERC20.balanceOf(address) (contracts/UnoERC20.sol#77-79)

Recommendation:
Declare methods as external.

LOW | RESOLVED

Emit event

It makes sense to emit an event for history and also to inform the front-end about some important settings changing.
contracts/Cohort.sol#119
contracts/Cohort.sol#275

Recommendation:
Emit event on important settings changing.

LOW | RESOLVED

Method is never used

The method is never used.
TransferHelper.safeApprove(contracts/libraries/TransferHelper.sol#6-14)

Recommendation:
Remove unused code.

LOW | RESOLVED

Zero checks

It's a good practice to use require(newValue != address(0), "ZERO_ADDRESS"); check before setting the storage address variable.
contracts/Cohort.sol#61
contracts/Cohort.sol#64
contracts/Cohort.sol#95
contracts/Cohort.sol#95
contracts/PremiumPool.sol#25
contracts/PremiumPool.sol#26

Recommendation:
Add the require statement.

LOW | UNRESOLVED

Use enumerableSet

There is an efficient Library which to search/find/delete for O(1) time
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/structs/EnumerableSet.sol
Use it here, instead of iterating over arrays.
contracts/Actuary.sol:10
contracts/Cohort.sol:39
contracts/Cohort.sol:42

Recommendation:
Use enumerableSet.

LOW | RESOLVED

Use ReentrancyGuard instead of lock

You have implementation of the lock modifier
contracts/Cohort.sol:69
It's better to write less code and import external well tested solutions e.g. from OpenZeppelin.

Recommendation:
Use
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/security/ReentrancyGuard.sol

LOW | RESOLVED

Max integer

contracts/Cohort.sol:48
Easy to make a mistake with the number of ‘f’. Also not readable.

Recommendation:
Use type(uin256).max

LOW | RESOLVED

Place the statement inside the unchecked section

At
contracts/UnoERC20.sol:261
You can place the statement one line above to the unchecked section.

Recommendation:
Save gas, place the statement unchecked.

LOW | RESOLVED

Place transfer in the balance-changing method

At contracts/PremiumPool.sol:51
It’s confusing that no transfer happens, but the balance is changed. Also opens the windows for a bug in future.

Recommendation:
Place the line from
contracts/Cohort.sol:146
To
contracts/PremiumPool.sol:51

LOW | RESOLVED

Why we should remain minimum amount in premium pool

It's not clear why do we need
contracts/Cohort.sol:203
More explanations are required

Recommendation:
Add comments.

LOW | RESOLVED

Why createPremiumPool not inside constructor

At
contracts/Cohort.sol:89
The method is always called after the constructor, why not just include the method inside the constructor.

Recommendation:
Move the logic to the constructor or add explanations in the comment in the source code.

LOW | RESOLVED

Gas optimisations in struct

At
contracts/Cohort.sol:15
The fields packing inside the struct is not really gas efficient.
It's better to place all uint256 vars at the top and the packing others by 256 slots.

Recommendation:
Change the order inside the structure to save the gas.

LOW | RESOLVED

Immutable variable

At
contracts/Cohort.sol:37
The ‘immutable’ modifier could be used. To declare it immutable and also to save gas.

Recommendation:
Use ‘immutable’ modifier.

LOW | RESOLVED

Resolve TODO

At
contracts/ClaimAssessor.sol:17
You have unresolved TODO:
//TODO should we use Off chain oracle?

Recommendation:
Resolve TODO.

LOW | RESOLVED

Unclear contract purpose

Lack of comments here
contracts/ClaimAssessor.sol
The purpose of the contract is not really clear.

Recommendation:
Add more comments to the code.

LOW | RESOLVED

Limit the array length

It's important to note that at
contracts/Cohort.sol#171-186
contracts/Cohort.sol#192-230
contracts/Cohort.sol#232-242
contracts/Cohort.sol#244-261
There are iterations over arrays. It consumes as much gas as the array length has. So starting from some point it will not be possible because of the max gas transaction limitation.

Recommendation:
Make a comment about maximum possible array length and use require to make sure the contract will not be misused.

LOW | RESOLVED

Potential reentry

The place looks potentially dangerous for reentry.
- TransferHelper.safeTransferFrom(token,_from,_pool,_amount) (contracts/Cohort.sol#160)
- IRiskPool(_pool).enter(_from,_amount) (contracts/Cohort.sol#161)
State variables written after the call(s):
- poolCapital[_pool] = _amount (contracts/Cohort.sol#162)
- poolCapital[_pool] += _amount (contracts/Cohort.sol#162)

Recommendation:
Use nonReentrant modifier.

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

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

	UnoERC20.sol	RiskPool.sol	RiskPoolFactory.sol
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Delegatecall Unexpected Ether	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions / Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

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

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

Contract: **Actuary**
- ✓ Should create Cohort by Cohort creator
- ✓ Should not allow other users to create cohort except cohort creator
- ✓ Should add one cohort creator
- ✓ Should add one cohort creator many times fails
- ✓ Should withdraw cohort creation fee from Actuary incorrect fee
- ✓ Should withdraw cohort creation fee from Actuary isCohortCreator
- ✓ Should withdraw cohort creation fee from Actuary

Contract: **ClaimAssessor**
- ✓ Should not allow others to request claim except owner

Contract: **Cohort**
- ✓ RiskPool leave
- ✓ RiskPool onlyCohort

Cohort Basic
- ✓ Should not allow others to add protocol
- ✓ Should add protocol
- ✓ Should deposit premium at cohort
- ✓ Should not allow others to create risk pool
- ✓ setDuration
- ✓ transferPremium
- ✓ Should create Risk Pool
- ✓ Leave require failed

Cohort Actions
- ✓ changePoolPriority

Cohort Staking
- ✓ Staking is Ended
- ✓ RiskPool overflow
- ✓ Leave from pool RiskPool not exist or empty
- ✓ Should enter in pool but cohort can be not started yet
- ✓ Should enter in pool and cohort was started

Cohort Claim Request & Withdraw
- ✓ Should not allow others to claim except claimAssessor
- ✓ Sender not factory
- ✓ Should claim revert with insufficient amount
- ✓ Should Forbidden because of duration
- ✓ Should request claim and get amount
- ✓ Should request claim and get amount
- ✓ Should leave with expected amount

Checking formula
[dust] 0
- ✓ Should check premium reward formula

Contract: ERC20
- ✓ has 18 decimals

total supply
- ✓ returns the total amount of tokens

balanceOf
when the requested account has no tokens
- ✓ returns zero

when the requested account has some tokens
- ✓ returns the total amount of tokens

transfer
when the recipient is not the zero address
when the sender does not have enough balance
- ✓ reverts

when the sender transfers all balance
- ✓ transfers the requested amount
- ✓ emits a transfer event

when the sender transfers zero tokens
- ✓ transfers the requested amount
- ✓ emits a transfer event

when the recipient is the zero address
- ✓ reverts

transfer from
when the token owner is not the zero address
when the recipient is not the zero address
when the spender has enough approved balance
when the token owner has enough balance
✓ transfers the requested amount
✓ decreases the spender allowance
✓ emits a transfer event
✓ emits an approval event
when the token owner does not have enough balance
✓ reverts
when the spender does not have enough approved balance
when the token owner has enough balance
✓ reverts
when the token owner does not have enough balance
✓ reverts
when the recipient is the zero address
✓ reverts
when the token owner is the zero address
✓ reverts
approve
when the spender is not the zero address
when the sender has enough balance
✓ emits an approval event
when there was no approved amount before
✓ approves the requested amount
when the spender had an approved amount
✓ approves the requested amount and replaces the previous one
when the sender does not have enough balance
✓ emits an approval event
when there was no approved amount before
✓ approves the requested amount
when the spender had an approved amount
✓ approves the requested amount and replaces the previous one
when the spender is the zero address
✓ reverts
decrease allowance
when the spender is not the zero address
when the sender has enough balance
when there was no approved amount before
✓ reverts

```
    when the spender had an approved amount
      ✓ emits an approval event
      ✓ decreases the spender allowance subtracting the requested amount
      ✓ sets the allowance to zero when all allowance is removed
      ✓ reverts when more than the full allowance is removed
    when the sender does not have enough balance
        when there was no approved amount before
      ✓ reverts
    when the spender had an approved amount
      ✓ emits an approval event
      ✓ decreases the spender allowance subtracting the requested amount
      ✓ sets the allowance to zero when all allowance is removed
      ✓ reverts when more than the full allowance is removed
when the spender is the zero address
      ✓ reverts
increase allowance
    when the spender is not the zero address
      when the sender has enough balance
      ✓ emits an approval event
when there was no approved amount before
      ✓ approves the requested amount
when the spender had an approved amount
      ✓ increases the spender allowance adding the requested amount
when the sender does not have enough balance
      ✓ emits an approval event
when there was no approved amount before
      ✓ approves the requested amount
when the spender had an approved amount
      ✓ increases the spender allowance adding the requested amount

when the spender is the zero address
      ✓ reverts
_mint
      ✓ rejects a null account
for a non zero account
      ✓ increments totalSupply
      ✓ increments recipient balance
      ✓ emits Transfer event
_burn
      ✓ rejects a null account
for a non zero account
      ✓ rejects burning more than balance
```

```
    for entire balance
      ✓ decrements totalSupply
      ✓ decrements initialHolder balance
      ✓ emits Transfer event
    for less amount than balance
      ✓ decrements totalSupply
      ✓ decrements initialHolder balance
      ✓ emits Transfer event
  _transfer
      when the recipient is not the zero address
        when the sender does not have enough balance
      ✓ reverts
when the sender transfers all balance
      ✓ transfers the requested amount
      ✓ emits a transfer event
when the sender transfers zero tokens
      ✓ transfers the requested amount
      ✓ emits a transfer event
when the recipient is the zero address
      ✓ reverts
when the sender is the zero address
      ✓ reverts
_approve
    when the spender is not the zero address
      when the sender has enough balance
      ✓ emits an approval event
when there was no approved amount before
      ✓ approves the requested amount

when the spender had an approved amount
      ✓ approves the requested amount and replaces the previous one
when the sender does not have enough balance
      ✓ emits an approval event
when there was no approved amount before
      ✓ approves the requested amount
when the spender had an approved amount
      ✓ approves the requested amount and replaces the previous one
when the spender is the zero address
      ✓ reverts
when the owner is the zero address
      ✓ reverts
104 passing (14s)
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	UNCOVERED LINES
contracts/	100.00	93.24	100.00	100.00	
Actuary.sol	100.00	100.00	100.00	100.00	
ClaimAssessor.sol	100.00	100.00	100.00	100.00	
Cohort.sol	100.00	91.18	100.00	100.00	
PremiumPool.sol	100.00	66.67	100.00	100.00	
RiskPool.sol	100.00	100.00	100.00	100.00	
UnoERC20.sol	100.00	100.00	100.00	100.00	
contracts/factories/	100.00	100.00	100.00	100.00	
CohortFactory.sol	100.00	100.00	100.00	100.00	
PremiumPoolFactory.sol	100.00	100.00	100.00	100.00	
RiskPoolFactory.sol	100.00	100.00	100.00	100.00	
All files	**100**	**100**	**100**	**100**	

We are grateful to have been given the opportunity to work with the Uno 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 Uno team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.