UNIMEX

SMART CONTRACT AUDIT

zokyo

June 27th 2022 | v. 1.0

Security Audit Score

PASS

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

TECHNICAL SUMMARY

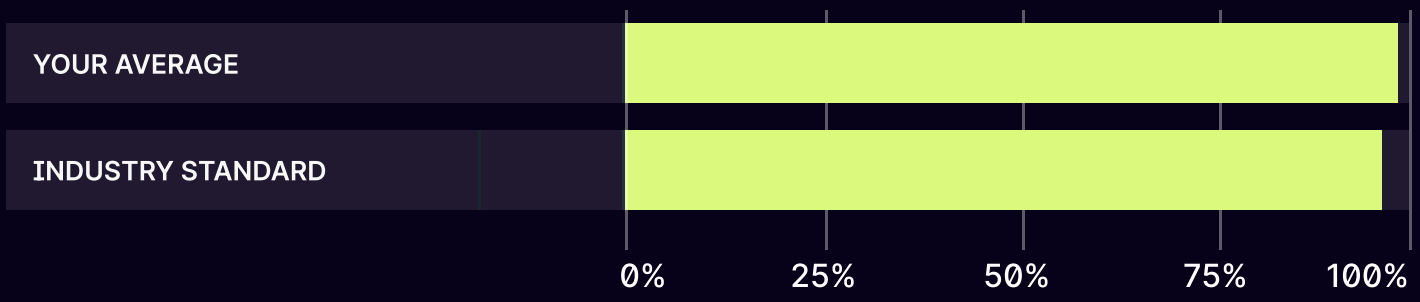
This document outlines the overall security of the Unimex smart contracts, evaluated by Zokyo's Blockchain Security team.

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

Contract Status

There were no critical issues found during the audit. (See Complete Analysis)

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 secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Unimex 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 Unimex repository.

Repository: https://github.com/wanglonghong/apemex-contracts

Last commit: zip.file

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

- UniMexMargin
- UniMexFactory
- SwapPathCreator
- PositionAmountChecker
- Supporting scope:
- UniMexPool
- UnimexConfig
- UniMexStaking
- UniMexToken
- Uniswap

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 Unimex 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:

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

Executive Summary

Contracts are well written and structured. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

There were no critical issues found during the audit. There were found issues with high and low severity and informational issues. Below you can see that not all issues were resolved by the Unimex team.

Based on the status of issues we can give a 93 security score.

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

HIGH | RESOLVED

In contract UniMexFactory.sol in functions setUtilizationScaled and setMaxLeverage the require checks at lines 45 and 51 are assigning a value instead of comparing it. For example at line 45 the value true is assigned to allowed. require(allowed[_token] = true);

Recommendation:
Change assignment to comparison.

HIGH | RESOLVED

In contract ApexMexMargin, function autoOpen, at line 462, there is no check to see if the transfer have been done successfully, there are some implementations of popular rc20 tokens that are using no require statements in the transfer operation and at the final are only using the return of an boolean value, the contract is already making use of SafeERC20 library and in other functions the method safeTransfer is used instead of transfer so we would recommend to implement the same pattern everywhere.

Recommendation:
Use safeTransfer method instead of transfer method.

LOW | UNRESOLVED

In contract UniMexStaking.sol, when a new token is set by function setToken, a number of issues may arise. A user's balance can be held for a token with a certain number of decimals, then another token with a different number of decimals can be set. Then the user's balance no longer correctly reflects how many tokens he has. Another problem that may arise would be that if a new token is set, the contract does not hold enough of that token for a user to use the withdraw function.

Recommendation:
Set the token address only through the constructor and avoid modifying it

INFORMATIONAL | UNRESOLVED

File ApemexMargin.sol contains the contract ApeMexMargin, there is a small type in the file name, it should be called ApeMexMargin, it is part of best practices to keep the file name and contract name the same.

Recommendation:
Rename the file ApemexMaring.sol to ApeMexMargin.sol

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, you are declaring the public variable uniswap_factory at line 96, and initializing with a value in the constructor at line 193, but there is no other use of it in the contract logic code.

Recommendation:
Remove the variable uniswap_factory as it is not used in the contract logic code.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, at line 94, variable staking is stored in contract state and is of type IUniMexMargin, this variable is used around in the contract logic code to make external calls to a contract, it would be more gas efficient to store the variable as an address type instead of type IUniMexMargin and to initialize the interface in place, because this way the interface will be stored in memory instead of storage and it is cheaper to read from memory.

Recommendation:
Change the type of variable to address, and initialize the interface inside the function when it is used.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, at line 95, variable unimex_factory is stored in contract state and is of type IUniMexFactory, this variable is used around in the contract logic code to make external calls to a contract, it would be more gas efficient to store the variable as an address type instead of type IUniMexFactory and to initialize the interface in place, because this way the interface will be stored in memory instead of storage and it is cheaper to read from memory.

Recommendation:
Change the type of variable to address, and initialize the interface inside the function when it is used.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, at line 97, variable uniswap_router is stored in contract state and is of type IUnswapV2Router02, this variable is used around in the contract logic code to make external calls to a contract, it would be more gas efficient to store the variable as an address type instead of type IUniswapv2Router02 and to initialize the interface in place, because this way the interface will be stored in memory instead of storage and it is cheaper to read from memory.

Recommendation:
Change the type of variable to address, and initialize the interface inside the function when it is used.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, at line 98, variable swapPathCreator is stored in contract state and is of type ISwapPathCreator, this variable is used around in the contract logic code to make external calls to a contract, it would be more gas efficient to store the variable as an address type instead of type ISwapPathCreator and to initialize the interface in place, because this way the interface will be stored in memory instead of storage and it is cheaper to read from memory.

Recommendation:
Change the type of variable to address, and initialize the interface inside the function when it is used.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, at line 95, variable unimexConfig is stored in contract state and is of type IUnimexConfig, this variable is used around in the contract logic code to make external calls to a contract, it would be more gas efficient to store the variable as an address type instead of type IUnimexConfig and to initialize the interface in place, because this way the interface will be stored in memory instead of storage and it is cheaper to read from memory.

Recommendation:
Change the type of variable to address, and initialize the interface inside the function when it is used.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, in function swapTokens, at line 829, you are initializing the IUniswapv2Router02 object using as parameter the uniswap_router variable, which is already saved in contract storage as type IUniswapRouter02 in conclusion is a casting to the same type, it is making the function more gas expense without any reasons because the casting to the same type it is not needed.

Recommendation:
If you chose to keep the same type of storage for uniswap_router remove the use of IUniswapRouter02 casting and just simply use the uniswap_router variable, if you chose to change the type of uniswap_router to addresses type, then leave it like that.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, the function setThresholdGasPrice, it's a setter that does not emit an event, it would be part of best practices to add events for all your setters to be able to track the change of values on-chain.

Recommendation:
Emit event for setter functions.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, the function setBorrowPercent, it's a setter that does not emit an event, it would be part of best practices to add events for all your setters to be able to track the change of values on-chain.

Recommendation:
Emit event for setter functions.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, the function setAmountThresholds, it's a setter that does not emit an event, it would be part of best practices to add events for all your setters to be able to track the change of values on-chain.

Recommendation:
Emit event for setter functions.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, the function setSwapPathCreator, it's a setter that does not emit an event, it would be part of best practices to add events for all your setters to be able to track the change of values on-chain.

Recommendation:
Emit event for setter functions.

INFORMATIONAL | UNRESOLVED

In contract ApeMexMargin, the function setFeesAddresses, it's a setter that does not emit an event, it would be part of best practices to add events for all your setters to be able to track the change of values on-chain.

Recommendation:
Emit event for setter functions.

INFORMATIONAL | UNRESOLVED

In contract ApexMexMargin, between lines 895-906, there are two functions named updateUniswapRouter and updateUniswapFactory, we would recommend to either delete them or uncomment them.

Recommendation:
Delete or make use of the commented functions.

INFORMATIONAL | UNRESOLVED

In contract UniMexStaking.sol at line 49, inside the constructor projectDivsDistributorAddress variable is initialized. The ProjectDivsDistributor.sol file has no implemented methods.

Recommendation:
If the ProjectDivsDistributor won't have any implementation, then simply remove the projectDivsDistributorAddress initialization.

INFORMATIONAL | UNRESOLVED

In contract UniMexStaking.sol WETH is initialized as IERC20 inside the constructor at line 47, then stored and used throughout the contract.

Recommendation:
In order to save gas, consider storing the WETH address and use in-place initialization when calling IERC20 methods, similar to how you use the token. Only the address is stored and then initialized when needed e.g at line 143.

INFORMATIONAL | UNRESOLVED

In contract ProjectDivsDistributor, in function distributeToken, there can be an optimization on saving gas by saving the value of addresses.length in a value in memory, because every time the for loop will read the value it will read it from storage, but if you made that modification, the value will be readed from memory and it is cheaper to read from memory then from storage:

Recommendation:
Save the value of addresses.length in a variable that is in memory and replace the use of addresses.length with the variable that is stored in memory.

INFORMATIONAL | UNRESOLVED

In contract ProjectDivsDistributor, in function distributeWei, there can be an optimization on saving gas by saving the value of addresses.length in a value in memory, because every time the for loop will read the value it will read it from storage, but if you made that modification, the value will be readed from memory and it is cheaper to read from memory then from storage:

Recommendation:
Save the value of addresses.length in a variable that is in memory and replace the use of addresses.length with the variable that is stored in memory.

INFORMATIONAL | UNRESOLVED

In contract UniMexStaking.sol in function _transfer at lines 185-201, there are no checks for the _from and _to address parameters.

Recommendation:
Add require checks to ensure those addresses are not the zero address.

INFORMATIONAL | UNRESOLVED

In contract UniMexToken.sol in function transfer at line 39 there's no revert message.

Recommendation:
Add a revert message.

INFORMATIONAL | UNRESOLVED

n contract UniMexToken.sol in function transfer at line 46 there's no revert message.

Recommendation:
Add a revert message.

INFORMATIONAL | UNRESOLVED

In contract PositionAmountCheck.sol in constructor at line 16 IUniswapFactory is initialized at an address that's not checked to not be zero.

Recommendation:
Add a check for address not zero.

INFORMATIONAL | UNRESOLVED

In contract PositionAmountCheck.sol in function setAmountThresholds at line 38, the parameter has a typo.

Recommendation:
Remove the 5 suffix in parameter 'leverage5' or if this was intended and the '5' has a meaning consider adding a comment.

INFORMATIONAL | UNRESOLVED

In contract UniMexConfig.sol inside the constructor at line 17 IUniMexFactory is initialized at an address that's not checked to not be zero.

Recommendation:
Add a check for address not zero.

INFORMATIONAL | UNRESOLVED

In contract UniMexFactory.sol in function setMarginAllowed at line 40 there's no revert message in the require check.

Recommendation:
Add a revert message.

INFORMATIONAL | UNRESOLVED

In contract UniMexFactory.sol in function setUtilizationScaled at lines 45 and 46 there are no revert messages in the require checks. Also, the function has a return of type uint256 but no return path.
Valid also for function setMaxLeverage, at lines 51 and 52, where there's no revert messages. The function also has no return path.

Recommendation:
Add revert messages for the require checks. Add return path, named return variable or remove the return type if it should be ignored.

In contract UniMexPool.sol in function initialize at lines 58 and 59 the pool token and WETH are being initialized and stored then used throughout the contract. This means the variable is loaded when it's used.

Recommendation:
In order to save gas, consider storing the addresses and use in-place initialization when calling `IERC20` methods i.e `IERC20(tokenAddress).some_method(..)`.

	ApeiMexMargin	UniMexFactory
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

	SwapPathCreator	PositionAmountChecker
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

	UnimexConfig	Uniswap
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

	UniMexPool	UniMexStaking
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

UniMexToken	
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 are written by Zokyo Secured team

As part of our work assisting Unimex 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 Unimex contract requirements for details about issuance amounts and how the system handles these.

PositionAmountChecker
- ✓ Should be deployed accurately
- ✓ Should be able to set amount thresholds
- ✓ Should be able to check position amount

ProjectDivsDistributor
- ✓ Should be deployed accurately
- ✓ Should set distribution and distribute (45ms)

SwapPathCreator
- ✓ Should be deployed accurately
- ✓ Should be able to set path (45ms)
- ✓ Should be able to get path
- ✓ Should be able to calculate converted value

UniMexFactory
- ✓ should set max leverage, all checks
- ✓ should add pool, all checks
- ✓ should set utilization scaled, all checks
- ✓ should set margin allowed, all checks
- ✓ should create pool, all checks

ApeMexMargin
- ✓ Should allow deposits
- ✓ Should allow withdrawals (57ms)
- ✓ Should be able to set amount thresholds
- ✓ Should be able to set Borrow Percent (55ms)
- ✓ Should be able to pause and unpause (70ms)
- ✓ Should be able to set staking (50ms)
- ✓ Should be able to set threshold gas price (66ms)
- ✓ Should be able to calculateAutoOpenBonus
- ✓ Should be able to calculateAutoCloseBonus (41ms)
- ✓ Should be able to open short position (358ms)

- ✓ Should be able to open long position (313ms)
- ✓ Should be able to close long position (285ms)
- ✓ Should be able to add commitmentToPosition (235ms)
- ✓ Should be able to set stop loss (264ms)
- ✓ Should be able to set take profit (255ms)
- ✓ Should be able to close position (442ms)
- ✓ Should be able to open limit order (136ms)
- ✓ Should be able to cancel limit order (94ms)
- ✓ Should be autoopen-ed (536ms)
- ✓ Should be able to open short position with sltp (539ms)
- ✓ Should be able to open long position with sltp (257ms)
- ✓ should set fees addresses, all checks (70ms)
- ✓ should add to position, all checks (511ms)
- ✓ should remove from position, all checks (487ms)
- ✓ should liquidate position, all checks (545ms)
- ✓ should close short position, all checks (741ms)
- ✓ it shoud close long all checks (619ms)

UniMexPool

- ✓ should distribute correction
- ✓ should withdraw tokens, all checks
- ✓ should claim tokens
- ✓ should transfer, all checks
- ✓ should check balanceOf
- ✓ should check corrected balance

UniMexStaking

- ✓ should set tokens, all checks
- ✓ should check total tokens supply
- ✓ should set projects divs, all checks
- ✓ should distr dividends
- ✓ should deposit
- ✓ should deposit from
- ✓ should claim
- ✓ should reinvest
- ✓ should check balanceOf

UniMexToken

- ✓ should distribute correction
- ✓ should withdraw tokens, all checks
- ✓ should claim tokens
- ✓ should transfer, all checks
- ✓ should check balanceOf
- ✓ should check corrected balance

Uniswap

```
    ✓ Should be able to sort tokens
    ✓ Should be able to get token output
    ✓ Should be able to get token pair
    ✓ Should be able to get reserves
    ✓ Should be able to get amounts out
  67 passing (33s)
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES
ApeiMexMargin	100	100	100	100
UniMexFactory	100	100	100	100
SwapPathCreator	100	100	100	100
PositionAmountChecker	100	100	100	100
UnimexConfig	100	100	100	100
Uniswap	100	100	100	100
UniMexPool	100	100	100	100
UniMexStaking	100	100	100	100
UniMexToken	100	100	100	100
All files	**100**	**100**	**100**	**100**

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

zokyo