WOMBEX

SMART CONTRACT AUDIT

zokyo

January 20th 2023 | v. 1.0

Security Audit Score

PASS

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

TECHNICAL SUMMARY

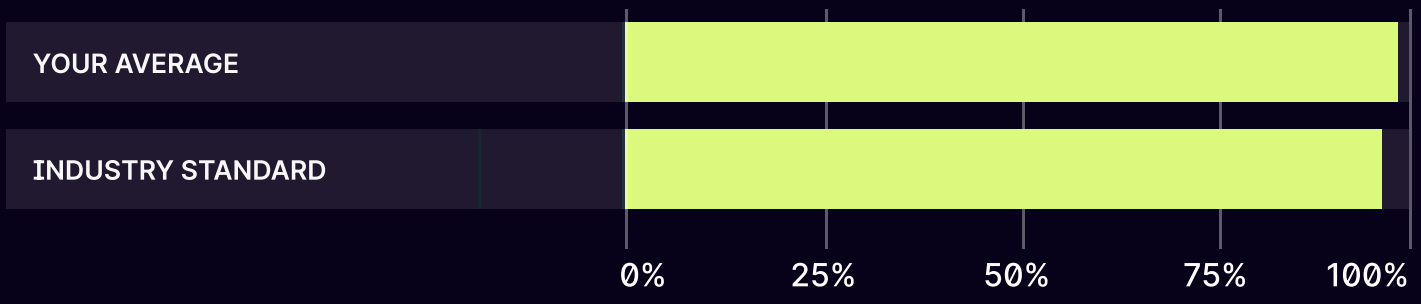
This document outlines the overall security of the Wombex smart contracts evaluated by the Zokyo Security team.

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

Contract Status

There were 0 critical issues found during the audit. (See Complete Analysis)

Testable Code



97.48% of the code is testable, 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 but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Wombex team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Wombex repository: https://github.com/wombex-finance/wombex-contracts

Last commit: 895e07b9490a37558036edb7149dbd387e43587b/contracts/WomDepositor.sol#L75

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- ExtraRewardsDistributor.sol
- Interfaces.sol
- Wmx.sol
- WmxClaimZap.sol
- WmxLocker.sol
- WmxMath.sol
- WmxMerkleDrop.sol
- WmxMinter.sol
- WmxPenaltyForwarder.sol
- WmxRewardPool.sol
- WmxVestedEscrow.sol
- WomDepositor.sol
- WomStakingProxy.sol
- BaseRewardPool.sol
- BaseRewardPool4626.sol
- Booster.sol
- CvxCrvToken.sol
- DepositToken.sol
- RewardFactory.sol
- TokenFactory.sol
- VoterProxy.sol
- BoringMath.sol
- IERC20Metadata.sol
- IERC4626.sol
- IGaugeController.sol
- IProxyFactory.sol
- IRewardHook.sol
- IRewarder.sol
- Interfaces.sol
- MathUtil.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Wombex smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat 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

There was no critical issue found during the audit alongside some issues with medium and low severity and two informational issues . All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. 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

Findings Summary

#	Title	Risk	Status
1	Return length of customLocksSlot of the account address from parameter instead of the sender	Medium	Resolved
2	Wrong check (Operator issue)	Low	Resolved
3	Refactor releaseLock return data type	Low	Resolved
4	Mix use of safeTransfer and transfer	Low	Resolved
5	SafeMath added in contracts compilex with solidity > 0.8	Low	Resolved
6	No confirmation support interface	Informational	Resolved
7	Deposited tokens never transferred to staking address if smartLockPeriod set to a very large value.	Informational	Resolved

MEDIUM | RESOLVED

Return length of customLocksSlot of the account address from parameter instead of the sender

In contract **WomDepositor**, the function `getCustomLockSlotsLength` takes an argument of type address. On inspecting the function signature, the function should return the length of slots for the given input address. But instead uses the address of the sender.

```
function getCustomLockSlotsLength(address _account) public view returns (uint256) {
   return customLockSlots[msg.sender].length;
}
```

Recommendation:

Fix the function logic as below

```
function getCustomLockSlotsLength(address _account) public view returns (uint256) {
    return customLockSlots[_account].length;
}
```

LOW | RESOLVED

Wrong check (Operator issue)

In contract WmxRewardPool,
The function `initialiseRewards` has wrong checks.
In this case, reward manager can invoke this method anytime (even if the calling time is too early for `startTime`).
Or if the calling time is in the future that `startTime`, anyone (not reward manager) can invoke this method.

```
require(msg.sender == rewardManager || block.timestamp > startTime,
"!authorized");
```

Recommendation:

Logical AND operator(&&) has to be used instead of Logical OR operator(||):

```
require(msg.sender == rewardManager && block.timestamp > startTime,
"!authorized");
```

or they have to be separated like this:

```
require(msg.sender == rewardManager, "!authorized");
require(block.timestamp > startTime, "too early");
```

LOW | RESOLVED

Refactor releaseLock return data type

In VoterProxy contract, releaseLock function is returning a boolean data type, however the function is only doing that because it is obligated by the interface, this is the only contract that is implementing the interface and also the return statement is not checked in the external call plus based on the function logic, it can only return true or revert, it has not case when it can return false.

Recommendation:
Refactor the releaseLock function from VoteProxy and from the interfaces where it is declared to not return a boolean anymore.

LOW | RESOLVED

Mix use of safeTransfer and transfer

There is a mix used of safeTransfer and transfer in the contract, the mix used is not taking into consideration if the token is made/controlled/deployed by the Wombex team or not, for a better context and extra safety it would be best to use safeTransfer for all the transfers.

Recommendation:
Use safeTransfers for all the transfers

LOW | RESOLVED

SafeMath added in contracts compilex with solidity > 0.8

Contract WomDepositor, WomStakingProxy, BaseRewardPool, Booster, CvxCrxToken, DespoitToken and VoterProxy are using SafeMath library, however this does not add any benefit because math since solidity 0.8 is checked, they are only using more gas, so they should be removed.

Recommendation:
Remove the use of SafeMath in all the contract mentioned above.

INFORMATIONAL | RESOLVED

No confirmation support interface

In contract WmxClainZap,
In the `constructor` there are no confirmations support interface for addresses.
The address of the `womdepositor, wmxWomRewards, locker` is immutable.

- `womdepositor` → `IWomDepositorWrapper`
- `wmxWomRewards` → `IBasicRewards`
- `locker` → `IWmxLocker`

If the provided addresses don't implement these interfaces, the contract will be deployed, but some transactions will be reverted.

```
womDepositor = _womDepositor;
wmxWomRewards = _wmxWomRewards;
locker = _locker;
```

Recommendation:

```
womDepositor = IWomDepositorWrapper(_womDepositor);
wmxWomRewards = IBasicRewards(_wmxWomRewards);
locker = IWmxLocker(_locker);
```

Deposited tokens never transferred to staking address if smartLockPeriod set to a very large value.

In contract **WomDepositor**, `smartLockPeriod` variable value can be set using the `setLockConfig` function, which only the owner can execute. If the owner acts with bad intent, `smartLockPeriod` can be set to a very high value, which affects the logic in ***_smartLock*** function:

```
if(lastLockAt + smartLockPeriod > block.timestamp && customLockDays[msg.sender] == 0) {
            return;
}
<else path never taken if smartLockPeriod is set too high>
<check if release is executed>
<calculate amount to transfer>
<transfer tokens to stake address>
<emit event SmartLock>
```

Here, `lastLockAt + smartLockPeriod` can be such a high value that **if** condition will always hold true and rest of the contract logic will never be executed.

Recommendation:
Restrict setting smartLockPeriod to only a certain range that is practical and acceptable, and this range should be hardcoded in the smart contract code.

	ExtraRewardsDistributor.sol Interfaces.sol Wmx.sol WmxClaimZap.sol WmxLocker.sol WmxMath.sol WmxMerkleDrop.sol WmxMinter.sol	WmxPenaltyForwarder.sol WmxRewardPool.sol WmxVestedEscrow.sol WomDepositor.sol WomStakingProxy.sol BaseRewardPool.sol BaseRewardPool4626.sol Booster.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

	CvxCrvToken.sol DepositToken.sol RewardFactory.sol TokenFactory.sol VoterProxy.sol BoringMath.sol IERC20Metadata.sol	IERC4626.sol IGaugeController.sol IProxyFactory.sol IRewardHook.sol IRewarder.sol Interfaces.sol MathUtil.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

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

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

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

```
    ✓ forward
    ✓ forward (114ms)
  BasePoolReward
    stake()
    ✓ stakes tokens for caller (99ms)
    stakeAll()
    ✓ stake all tokens balance of the transaction sender (74ms)
    stakeFor()
    ✓ stakes token for other address (85ms)
    withdraw()
    ✓ withdraws tokens staked (143ms)
    ✓ withdraws tokens staked (133ms)
    ✓ withdraws token staked on callers behalf (145ms)
    withdrawAll()
    ✓ withdraw all tokens staked (99ms)
    withdrawAndUnwrap()
    ✓ withdraw and Unwraps specified token amount (72ms)
    ✓ withdraw and Unwraps specified token amount (76ms)
    withdrawAllAndUnwrap()
    ✓ withdrawAllAndUnwrap() (83ms)
    queueNewRewards()
    ✓ queues rewards successfully when called by booster (267ms)
    queueNewRewards()
    ✓ queueNewRewards (399ms)
    ✓ queueNewRewards (266ms)
    donate()
    ✓ Donate rewards to contract (62ms)
    processIdleRewards()
    ✓ Processes queued rewards (280ms)
```

queueNewRewards()
✓ queueNewRewards (260ms)
lastTimeRewardApplicable()
✓ returns last Time Reward Applicable
rewardPerToken()
✓ gets reward Per Token for a token
earned()
✓ returns earned rewards by user

BaseRewardPool4626

deposit()
✓ deposit (48ms)
✓ deposit (51ms)
totalAssets
✓ totalAssets (121ms)
mint()
✓ mint (51ms)
withdraw()
✓ withdraw (81ms)
✓ withdraw (75ms)
approve()
✓ approve (54ms)
transfer()
✓ transfer (41ms)
decimals()
✓ decimals
symbol()
✓ symbol
totalSupply()
✓ totalSupply
name()
✓ name
redeem()
✓ redeem (139ms)

Booster

✓ withdrawTo() (951ms)
setOwner
✓ setOwner (50ms)
setFeeManager
✓ setFeeManager (54ms)

setPoolManager
✓ setPoolManager (54ms)
setFactories
✓ setFactories (97ms)
setExtraRewardsDistributor
✓ setExtraRewardsDistributor (148ms)
setRewardClaimedPenalty
✓ setRewardClaimedPenalty (77ms)
setVoteDelegate
✓ setVoteDelegate (54ms)
setVotingValid
✓ setVotingValid (54ms)
setLockRewardContracts
✓ setLockRewardContracts (122ms)
setMintRatio
✓ setMintRatio (58ms)
updateDistributionByTokens
✓ updateDistributionByTokens (93ms)
setEarmarkIncentive
✓ setEarmarkIncentive (76ms)
addPool()
✓ addPool (754ms)
deposit()
✓ Deposits amount to gauge and mints deposit tokens (949ms)
✓ earmarkRewards (847ms)
✓ earmarkRewards (3055ms)
✓ earmarkRewards (4293ms)
✓ withdrawAll() (1157ms)
shutdownPool()
✓ shutdownPool (875ms)
shutdownSystem()
✓ shutdownSystem (983ms)
depositAll()
✓ depositALl (937ms)
updateDistributionByTokens
✓ updateDistributionByTokens (287ms)
✓ updateDistributionByTokens (284ms)
rewardClaimed()
✓ rewardClaimed (3625ms)
✓ reclaimed locked (3080ms)
✓ reclaimed penalty (4267ms)
distributionByTokenLength()
✓ distributionByTokenLength (852ms)

setVote()
✓ sets vote (57ms)
voteExecute()
✓ Should execute vote (133ms)

CvxCrvToken

setOperator
✓ setOperator (62ms)
mint
✓ mint (57ms)
burn
✓ burn (119ms)

contract: CvxCrv

Initialization
✓ Should have correct initial values (109ms)
increaseAllowance()
✓ increases allowances for specified account (154ms)
decreaseAllowance()
✓ Should decrease allowances for account (116ms)
transfer()
✓ transfer to account specified amount of tokens (70ms)

DepositToken

burn
✓ burn (157ms)

contract: depositToken

Initialization
✓ Should have correct initial values (111ms)
increaseAllowance()
✓ increases allowances for specified account (148ms)
decreaseAllowance()
✓ Should decrease allowances for account (123ms)
transfer()
✓ transfer to account specified amount of tokens (99ms)

ExtraRewardDistributor

modifyWhitelist
✓ modifyWhitelist (61ms)

addReward()
✓ addReward (520ms)
addRewardToEpoch()

BigNumber { value: "1" }
✓ adds reward to epoch (164ms)
✓ addRewardToEpoch (106ms)
✓ addRewardToEpoch (675ms)

BigNumber { value: "1" }
✓ addRewardToEpoch (161ms)
claimableRewardsAtEpoch()
✓ claimableRewardsAtEpoch (179ms)
claimableRewards()
✓ claimableRewards (200ms)
forfeitRewards()
✓ forfeitRewards (1290ms)
✓ forfeitRewards past (394ms)
getReward()
✓ getReward (212ms)
getReward()
✓ getReward (724ms)
✓ getReward claimable (1237ms)

RewardFactory
CreateCrvRewards()
✓ CreateCrvRewards (313ms)

TokenManager
CreateDepositToken
✓ CreateDepositToken (134ms)

VoterProxy
setDepositor
✓ setDepositor (60ms)
setRewardDeposit
✓ setRewardDeposit (56ms)
setOperator
✓ setOperator
setLpTokensPid()
✓ setLpTokensPid() (259ms)

getName()
✓ getName() (85ms)
setVote()
✓ setVote() (141ms)
execute()
✓ execute (169ms)
withdrawLp()
✓ withdrawLp (218ms)
deposit()
✓ deposit (789ms)
✓ deposit/withdraw() (754ms)
✓ deposit/withdrawALl() (771ms)
✓ deposit/withdrawLp() (771ms)
✓ deposit() (463ms)
lock()
✓ lock (517ms)
claimCRV()
✓ CLAIMcrv (384ms)
setRewardDeposit()
✓ setRewardDeposit (46ms)

WmxClaimZap

getName()
✓ gets name
setApprovals()
✓ sets approvals for wom, wmx, womWmx tokens (636ms)
claimRewards()
wmxBalance: 0
✓ Claims all the rewards (1692ms)

Test Wmx

Should test init for Wmx.sol
✓ Call init (232ms)
✓ Call init with zero address at minter
✓ Update operator (126ms)
Should test mint for Wmx.sol
✓ Only minter can mint
✓ Call minter mint (62ms)
✓ Non-operator mints (58ms)
✓ Operator mints (67ms)
✓ Operator tries to mint without Wmx.sol intitialized (170ms)

Test mints

✓ Minter mints max2 (219ms)
✓ Operator mints max3 (334ms)
✓ Operator mints 0 (229ms)
✓ Minter mints max (239ms)

Test WmxMath

Test WmxMath

Should return min of two numbers

✓ Should return min of two numbers (46ms)

Should return sum of two numbers

✓ Should return v of two numbers (60ms)

Should subtaract two numbers

✓ 1 - 1 = 0 (58ms)

Should return product of two numbers

✓ 1 * 1 = 1 (59ms)

Should return div of two numbers

✓ 1 / 1 = 1 (52ms)

Should return average of two numbers

✓ avg(1, 1) = 1 (55ms)

Should convert uint type from uint256 to uint224

✓ Should return average of two numbers (60ms)
✓ Should revert due to overflow (91ms)
✓ Should allow max uint224 to be converted (55ms)

Should convert uint type from uint256 to uint128

✓ Should return average of two numbers (47ms)
✓ Should revert due to overflow (89ms)
✓ Should allow max uint128 to be converted (61ms)

Should convert uint type from uint256 to uint112

✓ Should return average of two numbers (58ms)
✓ Should revert due to overflow (95ms)
✓ Should allow max uint112 to be converted (59ms)

Should convert uint type from uint256 to uint96

✓ Should return average of two numbers (54ms)
✓ Should revert due to overflow (88ms)
✓ Should allow max uint96 to be converted (53ms)

Should convert uint type from uint256 to uint32

✓ Should return average of two numbers (43ms)
✓ Should revert due to overflow (83ms)
✓ Should allow max uint32 to be converted (60ms)

Test WmxMath32

Should return sub of two numbers

✓ Should return sub of two numbers (55ms)

Test WmxMath112

Should return sum of two numbers

✓ Should return sum of two numbers (50ms)

Should return sub of two numbers

✓ Should return sub of two numbers (55ms)

Test WmxMath224

Should return sum of two numbers

✓ Should return sum of two numbers (54ms)

Test WomDepositor

Should set configs for WomDepositor contract

✓ set lock config

✓ Should not allow non owner to set lock config (40ms)

✓ set custom Lock (40ms)

Deposit tokens without setting configs

✓ Should deposit user's tokens (1249ms)

✓ Should check if zero transfer is allowed (454ms)

✓ Should deposit without staking (820ms)

✓ Should deposit with lock configs set (417ms)

Deposit tokens with setting configs

✓ Should check deposit with custom deposit (554ms)

✓ Should deposit not trigger lock until lock period expires (923ms)

✓ Should trigger deposit lock after period expires (1016ms)

Deposit tokens with custom lock

✓ Should deposit user's tokens (495ms)

Test release lock funciton

✓ Should revert if slot is not ended (645ms)

✓ Should release slot after it ends (804ms)

✓ Should release slot after it ends (938ms)

Test WomStakingProxy

Test default values

✓ Should check default values (152ms)

Test contract functions

✓ Should update config values (181ms)

✓ Should set approvals (213ms)
Test contract rescueToken function
✓ Should allow rescuing tokens (273ms)
✓ Should fail to rescuse (102ms)
Test queueNewRewards
✓ Should queue new rewards with zero amount (155ms)
✓ Should fail to transfer due to insufficient balance (132ms)
✓ Should fail to transfer wom from user to contract due to insufficient allowance (193ms)
✓ Should transfer wom from user to contract (787ms)
✓ Should fail transfer wom from proxy to depositor due to allowance (291ms)
✓ Should distribute wmxWom to user (836ms)

WmxLocker

Operation without exceptions:
✓ #constructor - checks all initial config (154ms)
count: BigNumber { value: "2" }
✓ #lock, #userLocks, #lockDuration, #delegateeUnlocks - lock CVX (974ms)
✓ #delegate - supports delegation (499ms)
✓ #checkpointEpoch, #balanceAtEpochOf - checkpoint CVX locker epoch (7156ms)
count: BigNumber { value: "2" }
count: BigNumber { value: "3" }
✓ #addReward, #approveRewardDistributor, #queueNewRewards - notify rewards (848ms)
✓ #rewardPerToken, #rewardTokensLen, #getReward, #rewardTokens - get rewards from CVX locker (1690ms)
check delay: %S 0
expiry time: 1685610596
✓ #processExpiredLocks - process expired locks (1401ms)
✓ #setKickIncentive - set Kick Incentive (78ms)
✓ #recoverERC20 - recover ERC20 (469ms)
✓ #emergencyWithdraw - when user has locks (2326ms)
✓ #shotdown - shutdown the contract. unstake all tokens. release all locks (265ms)
#modifyBlacklist
✓ allows blacklisting of contracts (203ms)
✓ doesn't allow blacklisting of EOA's (76ms)
queueing new rewards
✓ distribute rewards from the booster (3205ms)
✓ queues rewards when wmxWom period is finished (3833ms)
✓ only starts distributing the rewards when the queued amount is over 83% of the remaining (11653ms)
Exceptions:
✓ #modifyBlacklist - Owner can transfer the ownership (43ms)
✓ #addReward - Owner can transfer the ownership (40ms)

✓ #approveRewardDistributor - Owner can transfer the ownership (125ms)
✓ #setKickIncentive - Owner can transfer the ownership
✓ #shutdown - Owner can transfer the ownership (39ms)
✓ #recoverERC20 - Owner can transfer the ownership (152ms)
✓ #lock - A malicious contract owner can shutdown the contract so that users can't lock tokens. (114ms)
✓ kickExpiredLocks() (181ms)
✓ kickExpiredLocks() (49ms)
✓ processExpiredLocks() (61ms)
✓ processExpiredLocks() (108ms)
✓ processExpiredLocks() (44ms)
✓ userLocksLen
✓ rewardTokensList()
✓ getPastTotalSupply() (192ms)

lockedBalances()

✓ lockedBalances

delegate()

✓ delegate (150ms)

getPastVotes()

✓ getPastVotes

addReward()

✓ addReward (52ms)

queueNewRewards()

✓ queueNewRewards (59ms)

emergencyWithdraw()

✓ emergencyWithdraw (54ms)

setKickIncentive()

✓ setKickIncentive (40ms)

recoverERC20()

✓ recoverERC20 (77ms)

approveRewardDistributor()

✓ approveRewardDistributor

getPastTotalSupply()

✓ getPastTotalSupply (48ms)

balanceAtEpochOf()

✓ balanceAtEpochOf

getReward()

✓ getReward

WmxMerkleDrop

Operation without exceptions:

✓ #constructor - initial configuration is correct (76ms)
✓ #claim - allows claiming and locking (176ms)

✓ #setDao - sets a new dao (48ms)
✓ #setRoot - sets a new root if it was not previously set (90ms)
✓ #rescueReward - rescue rewards (38ms)
✓ #startEarly - starts early the drop (43ms)
✓ #withdrawExpired - withdraw expired (350ms)
✓ #setLocker - set a new locker (283ms)

Exceptions:

✓ #setDao - abuse of the admin power(admin can set wrong address to new dao) (103ms)
✓ #setRoot - abuse of the admin power(admin can set wrong root to new root) (134ms)
✓ #setLocker - abuse of admin power(admin can set wrong address to new locker) (92ms)
✓ #withdrawExpired - abuse of the admin power(admin can set maximum expired time so that can't invoke) (71ms)
✓ #claim - WmxLocker is not checked (99ms)
✓ (91ms)
✓ (222ms)
✓ (282ms)

setDao

✓ setDao (241ms)

setRoot

✓ setRoot (226ms)

startEarly

✓ startEarly (214ms)

withdrawExpired

✓ withdrawExpired (209ms)

setLocker

✓ setLocker (209ms)

rescueReward

✓ rescueReward (279ms)

claim

✓ claim (66ms)

claim

✓ claim
✓ claim (251ms)

claim

✓ claim

WmxMinter

Operation without exceptions:

✓ #constructor - initial configuration is correct (46ms)
✓ #mint - mints tokens (402ms)

Exceptions:

✓ #mint - Admin power abuse. (385ms)

WmxPenaltyForwarder

Operation without exceptions:

✓ #constructor - initial configuration is correct (61ms)
✓ #forward - should forward all wmx balance to the distributor (2712ms)
✓ #setDistributor - sets the new distributor (267ms)

Exceptions:

✓ #setDistributor - abouse of the admin power (1352ms)

WmxRewardPool

Operation without exceptions:

✓ #constructor - initial configuration is correct (374ms)
✓ #stake - allows users to deposit before rewards are added (no rewards accrued) (735ms)
✓ #initialiseRewards - allows anyone to trigger rewards distribution after startTime (495ms)
✓ #getReward - accrues rewards to existing depositors following startTime (1790ms)
✓ #stake - allows subsequent deposits (596ms)
✓ #stakeFor - allows users to stake For someone else (873ms)
✓ #earned - penalises claimers who do not lock (799ms)
✓ #forwardPenalty - allows anyone to forward penalty on to the PenaltyForwarder (673ms)
✓ #forwardPenalty - only forwards penalties once (390ms)
✓ #stakeAll - allows users to stakeAll (986ms)
✓ #withdraw - allows users to withdraw (589ms)
✓ #withdraw - allows users to withdraw and claim rewards (1207ms)
✓ #withdraw - allows users to withdraw and stake rewards (2118ms)
✓ #rescueReward - rescues rewards before contract has started (14842ms)

Exceptions:

✓ #initialiseRewards - current time is earlier than startTime (74ms)
✓ #initialiseRewards - The caller is not reward manager (82ms)
✓ #withdraw - The malicious reward manager can change `wmxLocker` (668ms)
✓ #getReward - The malicious reward manager can change `wmxLocker` (2353ms)

WmxVestedEscrow

Operation without exceptions:

✓ #constructor - initial configuration is correct (92ms)
✓ #fund - fund recipients with rewardTokens (185ms)
✓ #available - get available amount to claim
✓ #remaining - get remaining vested amount
✓ #claim - claim reward token (106ms)
✓ #cancel - cancel recipients vesting rewardTokens (121ms)
✓ #setAdmin - change the contract admin (49ms)
✓ #setLocker - change the locker contract address (281ms)

Exceptions:
✓ #setAdmin - Abuse of the admin power. (62ms)
✓ #setLocker, #claim - Abuse of the admin power(transaction reverted due to admin change the locker address) (257ms)
✓ #cancel - Abuse of the admin power (152ms)
setAdmin
✓ setAdmin

✓ cancel (46ms)
fund
✓ fund
✓ fund
✓ fund

✓ (136ms)

284 passing (15m)

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
VoterProxy.sol	100	89.66	100	100	
TokenFactory.sol	100	100	100	100	
RewardFactory.sol	100	100	100	100	
DepositToken.sol	100	100	100	100	
CvxCrvToken.sol	100	100	100	100	
Booster.sol	99.01	86.46	100	98.53	
BaseRewardPool4626.sol	100	87.5	100	100	
BaseRewardPool.sol	97.22	83.33	100	97.25	
WomDepositor.sol	100	100	100	100	
ExtraRewardsDistributor.sol	94.34	86.36	100	94.34	
WmxVestedEscrow.sol	100	92.86	100	100	
WmxPenaltyForwarder.sol	100	100	100	100	
WmxMinter.sol	100	100	100	100	
Wmx.sol	100	100	100	100	
WmxClaimZap.sol	96.08	88.89	100	96.08	
WomStakingProxy.sol	100	100	100	100	
WmxLocker.sol	93.77	82.54	100	93.31	
WmxMath.sol	100	100	100	100	
WmxMerkleDrop.sol	100	92.11	100	100	
All files	**97.48**	**88.74**	**100**	**97.25**	

We are grateful for the opportunity to work with the Wombex team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Wombex team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo