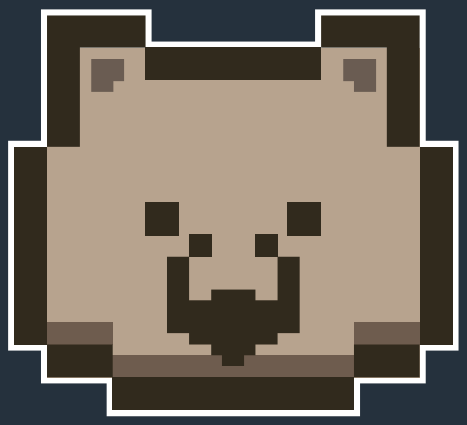


SMART CONTRACT AUDIT

ZOKYO.

20th April, 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.

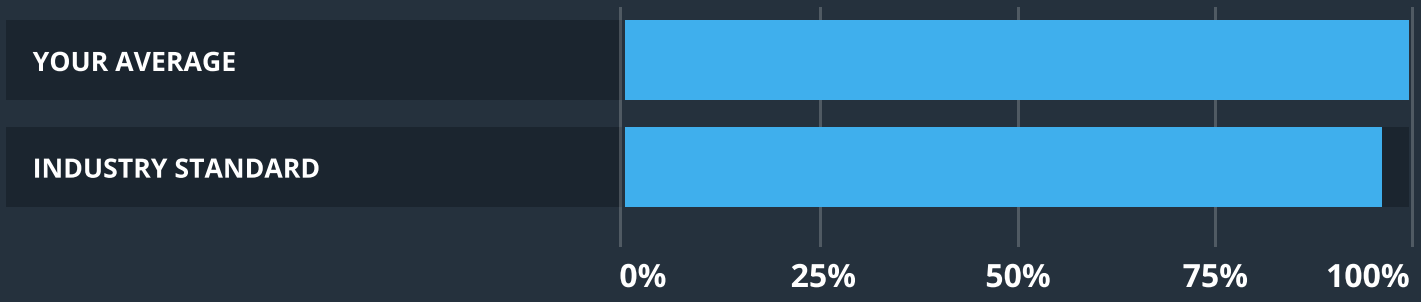
TECHNICAL SUMMARY

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

The scope of this audit was to analyze and document the Wombat Exchange 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 Wombat Exchange 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 Wombat Exchange repository.

Repository:
https://github.com/wombat-exchange/wombat

Last commit
d7c2e5af654bcc3051cf37fd8441d108d0d40889

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

- AggregateAccount
- Asset
- CoreV2
- TokenVesting
- PausableAsset
- Pool
- WombatERC20

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 Wombat Exchange 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.
2	Cross-comparison with other, similar smart contracts by industry leaders.
3	Testing contract logic against common and uncommon attack vectors.
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. 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

Recommendation:
In contract TokenVesting.sol, in the function setBeneficiary there are no checks in order to see that the WOM token balance of the contract is greater than or equal to the _totalAllocationBalance + allocation. This can be a centralization issue because the owner might choose not to send the tokens for a certain beneficiary. Also, it can lead to undefined behavior as some users might not claim on time, leading to them not being able to claim after others already claimed their share. Alternatively you could add a require for the contract balance to be greater than or equal to the new _totalAllocationBalance. This should be added before line 117 (before storing beneficiary info and increasing _totalAllocationBalance). Also consider adding a transferFrom directly in the function, after the beneficiary data has been stored, so that the setting and transferring is done in a single atomic transaction.

Recommendation:
In contract Asset.sol, override the “approve” function from ERC20, to not be used. Beware that changing an allowance with this method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering. Use IncreaseAllowance and DecreaseAllowance instead of Approve.

Recommendation:
In contract WombatERC20.sol, override the “approve” function from ERC20, to not be used. Beware that changing an allowance with this method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering. Use IncreaseAllowance and DecreaseAllowance instead of Approve.

LOW | RESOLVED

In contract DSMath.sol, at line 43, function wdiv is not consistent with the message in the comment above the function. The function does not return zero if x*y<WAD. Unexpected results may also occur when y has very low values.

INFORMATIONAL | RESOLVED

In contract Pool.sol, in function “setFeeTo”, error message that appears when “feeTo” is zero address is not the expected one.

Recommendation:
Use “_checkAddress” function instead of checking through an if, at line 238.

INFORMATIONAL | RESOLVED

Redundant cast in contract TokenVesting.sol, at line 143 timestamp is being cast to uint256 although it is received as a parameter of type uin256.

Recommendation:
Remove the cast at line 143.

INFORMATIONAL | RESOLVED

In contract TokenVesting.sol, redundant initialization of IERC20(vestedToken) at line 133 inside safeTransfer. The vestedToken is already an IERC20 initialized in the constructor.

Recommendation:
Remove the initialization of IERC20 inside the safeTransfer call. Also, might consider dropping the initialization in the constructor and storing only the address and initializing only inside the safeTransfer call at line 133, as it reduces gas.

INFORMATIONAL | RESOLVED

In contract TokenVesting.sol in function _vestingSchedule at lines 156-175 there is no default return path for the function. Also, the condition at line 165 will always be true as the parameter timestamp is passed from release function as block.timestamp.

Recommendation:
Consider changing the order of conditions by calling _calculateInterval only once and then have the isUnlocked check and update the _unlockIntervalsCount. Attached below is a snippet of how this can be refactored.

```
function _vestingSchedule(
    address beneficiary,
    uint256 totalAllocation,
    uint256 timestamp
) internal returns (uint256) {
    if (timestamp < start()) {
        return 0;
    } else if (timestamp > start() + duration()) {
        return totalAllocation;
    }

    uint256 currentInterval = _calculateInterval(timestamp);
    bool isUnlocked = currentInterval > beneficiaryInfo[beneficiary]._unlockIntervalsCount;
    if (isUnlocked) {
        beneficiaryInfo[beneficiary]._unlockIntervalsCount = currentInterval;
    }
    return (totalAllocation * currentInterval * 10) / 100;
}
```

INFORMATIONAL | RESOLVED

In contract Pool.sol, the state mutability of the functions at lines 114-132 can be restricted to pure as they operate on private variables and act as modifiers.

INFORMATIONAL | RESOLVED

In contract Pool.sol at lines 197-200 there's no event emitted after changing the dev address, similar to the Ownable approach which emits a OwnershipTransferred event

Recommendation:
Add an event for the setDev function.

INFORMATIONAL | RESOLVED

In contract Pool.sol the IMasterWombat is declared, line 63, as variable and set in storage through the setMasterWombat at lines 202-205. It is then used in function deposit, at lines 446 and 447.

Recommendation:
Consider declaring/storing only the address of the MasterWombat, as masterWombatAddress and use in-place interface initialization such as IMasterWombat(masterWombaAddress).someFunc(..) as this can reduce gas cost.

INFORMATIONAL | RESOLVED

In contract Pool.sol at lines 769-775 in function globalEquilCovRatio variables equilCovRatio and invariant are shadowed, leading to redeclaration in the function body.

Recommendation:
Recommendation: Either remove the named return types to avoid the shadowing and declare inside the function body or rename the invariant declaration at 770 and remove the uint256 redeclaration of equilCovRatio at 773.

INFORMATIONAL | RESOLVED

In contract Pool.sol at lines 348-350 function assetOf has a misleading naming. It receives a token address and returns the address of the IAsset.

Recommendation:
Consider renaming to addressOfAsset.

	AggregateAccount	Asset
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

	PausableAsset	CoreV2
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

	WombatERC20	Pool
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

	TokenVesting
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 the Wombat Exchange team

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

```
Contract: AggregateAccount
  [initial deploy]
    ✓ Should return correct deployed account name and type
  [setAccountName]
    ✓ Should revert if invoked by non-owners of contract
  [setAccountName]
    ✓ Should revert if empty name
  [setAccountName]
    ✓ Should return changed account name if invoked by contract owner
Testing Near Assert
    ✓ should detect variance 18 d.p
    ✓ should detect variance 6 d.p
Contract: Asset
  [initial deploy]
    ✓ Should return correct pool address
  [setPool]
    ✓ Should change the pool address
    ✓ Should revert as restricted to only owner
    ✓ Should revert as pool address cannot be zero
  [underlyingToken]
    ✓ Should return correct underlying token address
  [decimals]
    ✓ Should return correct decimals
  [cash]
    ✓ Should return correct cash balance
  [liability]
    ✓ Should return correct liability balance
  [underlyingTokenBalance]
```

✓ Should return 0 underlying token balance with initial deploy
✓ Should return 100 WAD underlying token balance after token transfer
[transferUnderlyingToken]
✓ Should transfer ERC20 underlyingToken from asset contract to user account
✓ Should revert as restricted to only pool
[mint]
✓ Should mint ERC20 Asset LP tokens
✓ Should revert as restricted to only pool
✓ Should revert when max supply is exceeded
✓ Should revert if Asset LP token transferFrom pool
✓ Should revert as invalid signature called for permitted allowance (15031ms)
✓ Should revert as sender transferFrom above permitted allowance
✓ Should transferFrom sender to pool if permitted allowance
[burn]
✓ Should burn ERC20 Asset LP tokens
✓ Should revert as restricted to only pool
[addCash]
✓ Should add cash amount to cash balance
✓ Should revert as restricted to only pool
[removeCash]
✓ Should remove cash amount from cash balance
✓ Should revert as cash balance cannot be negative
✓ Should revert as restricted to only pool
[addLiability]
✓ Should add liability amount to liability balance
✓ Should revert as restricted to only pool
[removeLiability]
✓ Should remove liability amount from liability balance
✓ Should revert as liability balance cannot be negative
✓ Should revert as restricted to only pool

Contract: CoreV2

[swapQuoteFunc] - the swap quote function
✓ Should return correct quote given initial variables
✓ Should return correct quote given very large amount swap (54ms)
✓ Should return very poor quote if input amount x > asset of token x by 2 times
[_coverageYFunc] - return the asset coverage ratio of token y ("Ry")
✓ Should return correct asset coverage ratio given initial variables
[_coefficientFunc] - return the quadratic equation b coefficient ("b")
✓ Should return correct quadratic equation b coefficient given initial variables
[_invariantFunc] - return the invariant constant ("D")
✓ Should return correct invariant constant between token x and y
[_convertToWAD] - return the token amount in WAD units

✓ Should return correct token amount for 8 decimal ERC20 token with 18 digit decimal precision
✓ Should return correct token amount for 24 decimal ERC20 token with 18 digit decimal precision
[_convertFromWAD] - return original token amount with correct decimal numbers
✓ Should return correct token amount for 8 decimal ERC20 token
✓ Should return correct token amount for 24 decimal ERC20 token
[depositRewardImpl]
✓ withdrawal - edge cases
[withdrawalAmountInEquilImpl] - return withdrawal amount in equil
withdrawal fee - edge cases
✓ 1
✓ 2
✓ 3
✓ 4
✓ 5
✓ 6
✓ 7
✓ 8
[exactDepositLiquidityInEquilImpl] - return exact deposit reward in equil
deposit - edge cases
✓ repro: liquidityToMint < amount
✓ 1
✓ 2
✓ 3
✓ 4
✓ 4

DSMath

✓ 566_893424.wmul(0.5 WAD) = 283_446712
✓ 100_000000.wdiv(200_000000) = 0.5 WAD
✓ (1 WAD).wmul(2 WAD) = 2 WAD
✓ (2 WAD).reciprocal() = 1/2 WAD
✓ (2 RAY).rpow(6) = 64 RAY

MasterWombat

Master Wombat Utils
✓ should pause and unpause (42ms)
✓ withdraw full wom balance on emergencyWithdraw
✓ only vewom can call updateFactor
✓ should set womPerSec correctly (1402ms)
✓ should revert if the same lpToken is added into the pool
✓ should set correct state variables womPerSec, vewom and emission partition (51ms)
✓ should check rewarder added and set properly (47ms)
✓ should allow emergency withdraw from MasterWombat (62ms)
✓ should give out woms only after farming time (156ms)

✓ should not distribute woms if no one deposit (111ms)
[USDC Pool] Base pool only
✓ should claim wom when withdraw (66ms)
[USDC Pool] boosted pool only
✓ should claim wom when withdraw (65ms)
[All pools] Base pool only
✓ should claim when withdraw
✓ should multiclaim from certain pools only (103ms)
✓ should multiclaim from one pool only
✓ should multiclaim from all pools (112ms)
✓ should claim when deposit
[All pools] Base + boosted pool
✓ should claim when withdraw (43ms)
✓ should claim when deposit (45ms)
[All pools] vewom integration test
✓ should set & update factor and sumOfFactors correctly (389ms)
[All pools] LP token Migration
✓ should revert if newMasterWombat not set
✓ user should be able to migrate once only (1513ms)
✓ should claim wom before migrate() (4188ms)
✓ should claim wom in both old and new MasterWombat pool during migrate() (2151ms)

Contract: PausableAssets

[requireAssetNotPaused] - makes a function callable only when the asset is not paused
✓ Should not revert when asset does not exist
✓ Should revert when asset is paused
[requireAssetPaused] - makes a function callable only when the asset is paused
✓ Should not revert when asset is paused
✓ Would revert when asset does not exist
✓ Should revert when asset is not paused (47ms)
[_pauseAsset] - triggers pause state
✓ Should pause an asset and emit a pause asset event
[_unpauseAsset] - returns to normal state
✓ Should unpause an asset and emit an unpause asset event

Contract: Pool - Deposit

Asset BUSD (18 decimals)
quotePotentialDeposit
✓ works
deposit
✓ works (first LP) (48ms)
✓ works (second LP) (101ms)
✓ maintains the LP token supply and liability ratio (112ms)
✓ reverts if passed deadline

✓ reverts if liquidity to mint is too small
✓ reverts if liquidity provider does not have enough balance
✓ reverts if pool paused
✓ reverts if asset paused
✓ reverts if pause asset is invoked by non-owner
✓ allows deposit if asset paused and unpaused after (71ms)
✓ reverts if zero address provided
✓ reverts if asset not exist (2097ms)

Asset vUSDC (8 decimals)

deposit

✓ works (first LP) (50ms)
✓ works (second LP) (105ms)
✓ maintains the LP token supply and liability ratio (108ms)

3 assets, r* = 1, A = 0.001

✓ rx = 0.80 (157ms)
✓ rx = 1.53 (154ms)
✓ A = 0.002, should handle rounding error (117ms)

deposit and stake

✓ should work (73ms)

Contract: Pool - Swap

Asset BUSD (18 decimals) and vUSDC (6 decimals)

swap

✓ works (BUSD -> vUSDC) without haircut fees (119ms)
✓ works (BUSD -> vUSDC) with haircut fees (113ms)
✓ works (vUSDC -> BUSD) without haircut fees (124ms)
✓ works (vUSDC -> BUSD) with haircut fees (119ms)
✓ works (BUSD -> exact vUSDC output) 18 and 8 decimals without haircut fees (111ms)
✓ works (BUSD -> exact vUSDC output) 18 and 8 decimals with haircut fees (101ms)
✓ works (BUSD -> exact USDT output) both 18 decimals with haircut fees (98ms)
✓ reverts if asset paused
✓ allows swap if asset paused and unpaused after (91ms)
✓ allows swapping then withdrawing
✓ reverts if passed deadline
✓ reverts if amount to receive is less than expected (81ms)
✓ reverts if pool paused
✓ reverts if zero address provided
✓ reverts if asset not exist (4054ms)
✓ reverts if cov ratio will be less than 1% (66ms)

Contract: Pool - Fee

Various Paths

✓ should not set fee to 0
✓ fee should not collected if retention ratio is 1 (240ms)

```
    r* = 1: Asset BUSD (18 decimals), vUSDC (6 decimals) and CAKE (18 decimals)
    single swap
      ✓ works (BUSD -> vUSDC) without haircut fees (142ms)
      ✓ works (vUSDC -> BUSD) with haircut fees and no dividend (216ms)
      ✓ works (BUSD -> vUSDC) with haircut fees and dividend (290ms)
      ✓ (BUSD -> vUSDC) should respect mintFeeThreshold (257ms)
      ✓ works (BUSD -> vUSDC) with haircut fees, dividend and LP dividend (372ms)
      ✓ works (vUSDC -> BUSD) with haircut fees and dividend + deposit to mint fee (274ms)
      ✓ works (vUSDC -> BUSD) with haircut fees and dividend + withdraw to mint fee (300ms)
    multiple swap
      ✓ works and collect fee (587ms)
    r* = 1: 3 assets
    multiple swap
      ✓ A = 0.001 and collect fee (422ms)
Contract: Pool - Utils
    Get and set params, haircut and retention ratio
      ✓ Should get and set correct params (40ms)
      ✓ Should revert if notOwner sets contract private parameters
      ✓ Should revert if retention + lp dividend > 1
      ✓ Should revert if params are set outside out of their boundaries
    Add and configure Assets BUSD, USDC, and USDT
    Add ERC20 Asset
      ✓ works
      ✓ reverts for invalid params
      ✓ restricts to only owner
    Remove ERC20 Asset
      ✓ works (39ms)
      ✓ reverts for invalid params
      ✓ restricts to only owner
    addressOfAsset
      ✓ returns the address of asset
    getters
      ✓ works
      ✓ can change pool dev
      ✓ get tokens
    pausable
      ✓ works
      ✓ estricts to only dev (deployer)
    fillPool
      ✓ should revert if not enough value in tip bucket (175ms)
      ✓ should work (168ms)
Contract: Pool - Withdraw
```

Asset BUSD (18 decimals)
withdraw
✓ works (first LP) (94ms)
✓ works to withdraw all (84ms)
✓ reverts if passed deadline
✓ reverts if liquidity provider does not have enough liquidity token
✓ reverts if amount to receive is less than expected (65ms)
✓ reverts if no liability to burn
✓ reverts if pool paused
✓ reverts if zero address provided
✓ reverts if asset not exist (1295ms)
withdrawFromOtherAsset
✓ reverts when withdraw all liquidity (148ms)
✓ reverts when deadline passes
✓ reverts when amount is too low (84ms)
✓ reverts when pool is paused
✓ reverts when toToken is paused
✓ works when fromToken is paused (97ms)
✓ withdraw token0 from token1 works (171ms)
✓ withdraw more token0 than available
withdraw more token0 than available multiple times
✓ (10)
✓ (5, 5)
✓ (1, 9)
✓ (0.1, 9.9)
✓ (9, 1)
✓ (9.9, 0.1)
✓ (3, 3, 3, 1)
quotePotentialWithdraw
✓ works with fee (53ms)
✓ works with 0 fee (cov >= 1)
Asset vUSDC (8 decimals)
withdraw
✓ works (first LP) (89ms)
3 assets
✓ r* = 1, r = 0.8, withdraw fee > 0 (161ms)
✓ r* = 1, r = 1.7, A = 0.001, withdraw fee > 0 (169ms)

Contract: Asset (proxy)
deploy
Warning: Potentially unsafe deployment of Pool

```
You are using the `unsafeAllow.delegatecall` flag.
    ✓ should initialize correctly
upgrade
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
    ✓ should keep storage correctly (44ms)
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
    ✓ multiple upgrade should success (44ms)
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
    ✓ should not change assets (51ms)
    ✓ change admin
Warning: Potentially unsafe deployment of Pool
You are using the `unsafeAllow.delegatecall` flag.
Warning: Potentially unsafe deployment of TestPoolV2
You are using the `unsafeAllow.delegatecall` flag.
    ✓ should change implementation address (2881ms)
SignedSafeMath
[add] - adds 2 integers
    ✓ 20000 + 33000 = 53000
    ✓ 2 * 10**18 * WAD + 3.3 * 10**18 * WAD = 5.3 * 10**18 * WAD
    ✓ 20000.2 WAD + 33000.3 WAD = 53000.5 WAD
    ✓ 1.2 + 3.3 = throw underflow error
[sub] - subtracts 2 integers
    ✓ 53000 - 33000 = 20000
    ✓ 53000 WAD - 73000 WAD = -20000 WAD
[mul] - multiplies 2 integers
    ✓ 2.2 WAD * 4 = 8.8 WAD
    ✓ 2.2 WAD * -4 = -8.8 WAD
    ✓ 2.2 * 10**18 WAD * 4 WAD = 8.8 * 10**36 * WAD
[div] - divides 2 integers
```

✓ 8.8 WAD / 4 = 2.2 WAD
✓ 8.8 WAD / -4 = -2.2 WAD
[sqrt] - square roots an integer
✓ sqrt(9 WAD) = 3 * 10**9
✓ sqrt(81) = 9
✓ (-9 WAD) = 1

Contract: TokenVesting
[initial deploy]
✓ Should return correct start timestamp
✓ Should return correct vesting duration
✓ Should return 0 underlying WOM token balance
✓ Should return 0 beneficiary count
✓ Should return 0 total allocation balance
✓ Should return 0 released amount for user1
✓ Should return 0 vested amount for user1
[setBeneficiary]
✓ Should set new beneficiary address with allocation amount
✓ Should set 2 new beneficiary address with allocation amount
✓ Should revert set new beneficiary address if already set
✓ Should revert set new beneficiary address if not called by owner
[vestedAmount]
✓ Should calculate the correct amount of vested WOM tokens for a beneficiary (55ms)
✓ Should calculate the correct 8 decimal amounts of vested WOM tokens for a beneficiary (51ms)
✓ Should increment correct unlock interval count and transfer correct amount of vested WOM tokens (84ms)
✓ Should transfer correct amount of vested WOM tokens for 2 new beneficiary address after multiple interval counts (102ms)
✓ Should not return any amount of vested WOM tokens if claim before cliff

VeWOM
✓ should set correct name and symbol (46ms)
✓ should set MasterWombat correctly (48ms)
✓ should set NFT correctly
✓ should pause and unpause (63ms)
✓ should not allow deposit if not enough approve
✓ should not allow mint from smart contract unless whitelisted (6450ms)
✓ lock day should be valid (62ms)
✓ should allow minting multiple times with different length (193ms)
✓ lock 7 days (97ms)
✓ lock 1 years (123ms)
✓ lock 4 years (98ms)
✓ should respect maxBreedingLength (505ms)
✓ burn should work (263ms)

```
    ✓ burn should reject if time not reached yet (121ms)
    ✓ cannot stake nft if user has no wom staked
    ✓ stakes nft
Contract: WombatERC20
  [initial deploy]
    ✓ Should return correct name
    ✓ Should return correct symbol
    ✓ Should return correct decimals
    ✓ Should return correct total supply
    ✓ Should return correct balance of deployer
  [transferFrom with approve]
    ✓ Should revert as user has not approved transferFrom
    ✓ Should transferFrom deployer to user 1000 WOM tokens
  [transferFrom with increaseAllowance]
    ✓ Should revert as user transferFrom above allowance
    ✓ Should revert as user transferFrom above allowance, altered by decreaseAllowance
    ✓ Should transferFrom deployer to user 2000 WOM tokens
  [transferFrom with permit]
    ✓ Should revert as user transferFrom above permitted allowance (40ms)
    ✓ Should transferFrom sender to user 1000 WOM tokens (38ms)
  242 passing (39m)
  14 pending
```

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

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

- ✓ Should be able to be unpause an asset (45ms)
- ✓ Should be able to be set dev (61ms)
- ✓ Should be able to be set master wombat
- ✓ Should be able to be set AMP factor (56ms)
- ✓ Should be able to be set hair cut rate (55ms)
- ✓ Should be able to be set fee (89ms)
- ✓ Should be able to be to change the fee beneficiary (45ms)
- ✓ Should be able to set mint fee threshold
- ✓ Should be able to add assets (58ms)
- ✓ Should be able to remove assets (71ms)
- ✓ Should be able to deposit (228ms)
- ✓ Should be able to stake while depositing (329ms)
- ✓ Should be able to quote potential deposit (52ms)
- ✓ Should be able to withdraw (454ms)
- ✓ Should be able to quote withdrawal (199ms)
- ✓ Should be to withdraw from one asset to another (516ms)
- ✓ Should be able to swap (437ms)
- ✓ Should quote potential swap (559ms)
- ✓ Should be able to get exchange rate (144ms)
- ✓ Should be able to get global equil cov ratio (237ms)
- ✓ Should get tip bucket balance (489ms)
- ✓ Should allow dev fill pool (295ms)
- ✓ Should be able to transfer tipbucket (291ms)
- ✓ Should get asset corresponding to token (61ms)
- ✓ Should be able to mint fees (109ms)

Contract: **Token Vesting**

- ✓ Should be deployed correctly
- ✓ Should be able to set beneficiary (181ms)

```
    ✓ Should be able to get number of beneficiaries  (44ms)
    ✓ Should be able to get beneficiary balance  (167ms)
    ✓ Should get total allocation balance (92ms)
    ✓ Should  get total underlying balance (209ms)
    ✓ Should be able to get vested amount (2195ms)
    ✓ Should be able to calculate interval (259ms)
    ✓ Should be able to release vested tokens  (126ms)
    ✓ Should be able to get released tokens  (285ms)
  Contract: PausableAssets
    ✓ Should be able to pause assets (65ms)
    ✓ Should be able to unpause assets (79ms)
  Contract: WombatERC20
    ✓ Should be deployed correctly (224ms)
  69 passing (54s)
```

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	Uncovered Lines
AggregateAccount	100	100	100	100	
Asset	100	100	100	100	
CoreV2	100	100	100	100	
PausableAsset	100	100	100	100	
Pool	99.06	91.07	96.15	99.09	133, 849
WombatERC20	100	100	100	100	
TokenVesting	100	100	100	100	
All files	**99.4**	**95.1**	**97.94**	**99.41**	

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

ZOKYO.