Cyberscope

Audit Report

Dual Pools

January 2023

Github	https://github.com/JavisJL/dualpools
Commit	7855088b448a5b88973c36ce0fdc3bf5d09b0a27
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/JavisJL/dualpools
Commit	7855088b448a5b88973c36ce0fdc3bf5d09b0a27

Audit Updates

Initial Audit	19 Jan 2023

Source Files

Filename	SHA256
AggregatorV2V3Interface.sol	d1ddf377b603b138396ca9246e6ca0dd3ede629768d9d98c9c44520d1205e585
BEP20Interface.sol	5a126c0688e2a767cf9d14bd5c4bb922c50db92e4e62a622eeefd9dfb36be6fa
CarefulMath.sol	4d7f56d0ff01bb44ff9b6773bf55274574477c816753c22ddd34e658a296f900
Comptroller.sol	78844ca298cf7cac09971bb5c1b5fa4fe2bb1d05b77556a7dadff1e1353b637d
ComptrollerInterface.sol	9bb329b1d7031261da0d207ab6911cfd40fdce0406795868bc15759f42e3465a
ComptrollerLens.sol	29c41591c6504f839e16d6ac5d6822b008cbaf3b1c1752cbdbc70d930cfb9d29
ComptrollerLensInterface.sol	4a02bebdaf14280aa1fce2e6be6f21684479313b9168e7aa070fb624c68f514d
ComptrollerStorage.sol	40e19cbc46daf4b97293b98f1bbdd3ab6120a9115e40db3a79436b384ecad5e5
EIP20Interface.sol	3ee5bbdd464b6b96321cda70c0ee95f4c2676b9292da887814caca9d68da6c81
EIP20NonStandardInterface.sol	03f6818417f9209dc0902f52c2f46227a827a672ad5af0f16b2706e174c09de3
ErrorReporter.sol	9160aa851a926b5db8dad6bf62b27d5ff7459a23f5facf4738ae9c0e2a67658e
Exponential.sol	00e5b193661b1e003b620461b25565136ea936de83eaf7e6f1e0785a05d5ac27
ExponentialNoError.sol	6700b13c25c4240304a590fda5ab0c1fd5eadf764975e4e6d72ee87ad72643db

InterestRateModel.sol	e7a4beea855785e87adbc63a2e264c44043aa09c5866c94f6766d4ee1388f714
ITradeModel.sol	df0657410eb490ab5fa9e0a75257d8f73857c09975ff9cb51b3bfcaff2558421
JumpRateModel.sol	fa0e0eeb6b12a3a34ac2765a507801c9be72529f6c9f7ad705fb5a62c32d3f36
lib.sol	581e167c2bfcdd01484bc09f86f5f8f78c16a2c05525fb23011612bedc6258ce
PriceOracle.sol	fed698ff4b906f82baf23ca905243e01e3a6684363bc329dabf971076b416a5d
SafeMath.sol	4a47d15402f20ec26b0fe15d61f4f6e946e7949b7beaa6398957b5cadee42931
TradeModel.sol	81249993d7ee6b42c044c72702bb09e6a56d8cb3c19c113464136d7a82154e50
Unitroller.sol	bb18d95ec5f27d2179deb0d4c9ea8f6ebb02914dec41543a7895462d48c693a0
VAI.sol	1ce1f7718c6a0fe37f100d704aa68f74b353114f7cb038524ebc61b61cd19e50
VAIControllerInterface.sol	ddb382742c00daee01729fb122b57ea48e98474752b7fe414d0b405c81051c1c
VBep20.sol	fc7b0f626f050418722a5dad94112cc481d165520a73f11e7efa21ecbc7c191b
VBep20Delegate.sol	0b377e3eb1dd58a9c3b25e1e7e084427fa9463424e6eb75f591818aaf938795c
VBep20Delegator.sol	d236d708818da5ff559304c1c13e0345e0834bddfcfc9a1715ab05ce341e25b9
VBep20Immutable.sol	cf5e99a84923419b762abe7877d7c933b5bf8058dfbb753915a52f624c645738
VBNB.sol	8fd337efffa8a48efe349af356873683acf7dc4b9867262e80ff92ef3d1df366

VenusChainlinkOracle.sol	30da5aa12f904fb1d0aed035089bfe0fc8c2ca990843fe8f60d17544e77ba3a9
VToken.sol	8046339ce2465c3a42560dfc132f7f44a35bcb66130e19069b57d031d5b1d673
VTokenInterfaces.sol	8305c1eb700cf4a2167fce47eea1b20fae4baaa585fb6d63d2a5df82ce8e9047
XVS.sol	b0b7b4f2b6feafcece1be6bd55b77ca7d3c7d188993209796e7382606985129f

Introduction

DualPool implements a mechanism for supplying or borrowing assets. The users submit funds in order to receive vTokens or borrow funds (Cryptocurrency). The submitted funds are operating as collateral. The DualPool also provides a mechanism for trading the supported cryptocurrency with each other. The users have the ability to deposit one cryptocurrency in exchange for another cryptocurrency. The protocol implements a price mechanism that is based on the trade rate of each token.

DualPools is a Venus Protocol fork. This audit focuses on the changes that have been introduced by the DualPools team. The forked project has extended many segments of the Venus codebase. The files that have mainly affected/added are:

1. Comptroller.sol
2. VToken.sol
3. VBep20.sol
4. VBNB.sol
5. TradeModel.sol

Amount calculation

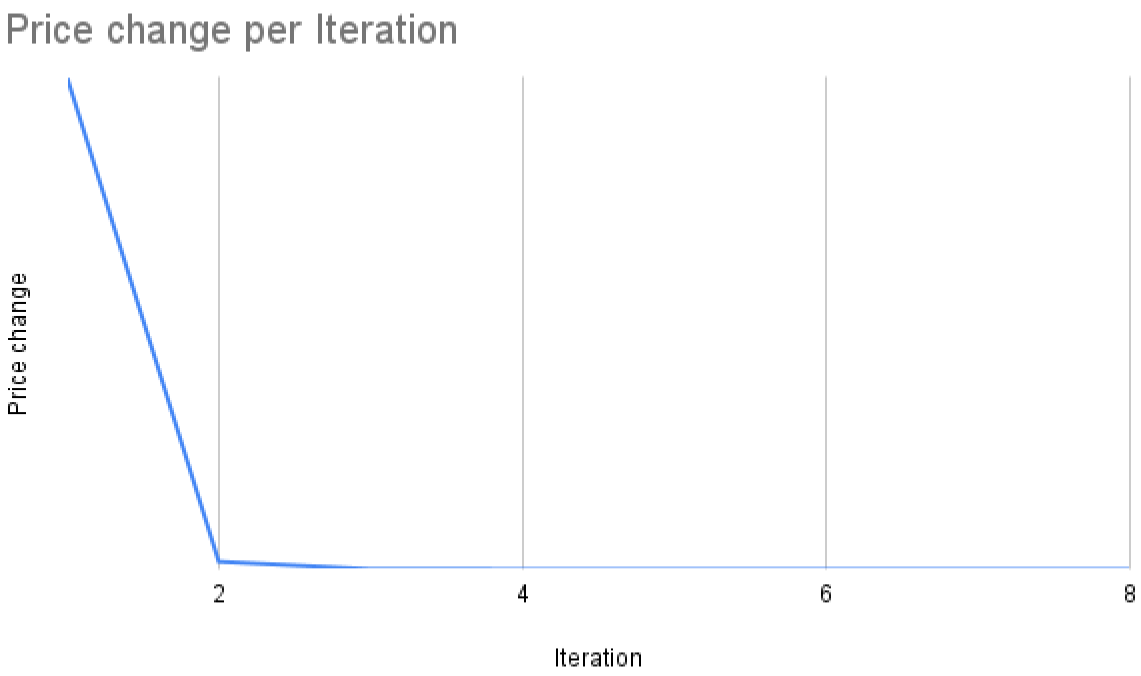
The DualPool implements a formula to evaluate the price of the underlying tokens based on the trading impact. The price is changed according to the trades similar to a classic DEX logic. According to the whitepaper, this is the price adjustment formula:

```
iUSDrate = iUSDbalance / (cash*oraclePrice + iUSDbalance)
Price impact = iUSDrate * abs(iUSDrate)
adjustedPrice = oraclePrice * (1 - abs(Price impact))
```

The implementation re-evaluates the adjustedPrice 3 times, providing the new price to the formula on every iteration. The following table depicts the price adjustment re-enforce on every iteration. The calculations are based on the variables `iUSDbalance = 1000; Cash = 10000; oraclePrice = 1;`

Iteration	Price	Change
1	0.9917355372	-
2	0.9916099393	0.0001266445889
3	0.9916080085	0.000001947126855
4	0.9916079788	0.00000002993807002
5	0.9916079783	0.0000000004603129449
6	0.9916079783	0
7	0.9916079783	0
8	0.9916079783	0

We observe that after the third/fourth iteration, the price change tends to zero. Thus it seems a good iteration threshold.



Swap Price Model

The swap feature of the DualPool trades two cryptocurrencies. It accepts one as an exchange for the other. The rate between the two cryptocurrencies depends on two variations.

1. The price of each cryptocurrency.
2. The taxed amount.

As we observe that the well-known decentralized exchange implementation, like Uniswap, the exchange is performed before the price adjustment. Thus, the users are aware of the price that they are going to trade. In the DualPool implementation, the price is adjusted prior to the exchange. We state that this may be the expected behavior of the DualPools business logic, but we mention the diversion with a classic swap mechanism.

https://github.com/Uniswap/v2-core/blob/master/contracts/UniswapV2Pair.sol

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🔴	RV	Reentrance Vulnerabilities	Unresolved
🔴	PAO	Potential Arithmetic Overflow	Unresolved
⚪	PTAI	Potential Transfer Amount Inconsistency	Unresolved
⚪	FAD	Fixed Address Deployment	Unresolved
⚪	PHI	Permissions Handling Inconsistency	Unresolved
⚪	L02	State Variables could be Declared Constant	Unresolved
⚪	L05	Unused State Variable	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L08	Tautology or Contradiction	Unresolved
⚪	L14	Uninitialized Variables in Local Scope	Unresolved

RV - Reentrance Vulnerabilities

Criticality	Critical
Status	Unresolved

Description

The VBnb contract is using a method `swapExactETHForTokens` that swaps native currency for a token via the DualPools mechanism. This method accepts BNB and optionally sends an amount to a potential referrer. The transfer of the amount is achieved via native currency transfer. The recipient could be a vulnerable contract that handles the reception of the payment. At this point, the paired token has not sent the corresponding amount to the initial issuer. As a result, the vulnerable contract knows that the price impact of the paired token is going to change. This is a scenario that depicts how the reentrance could be exploited by a malicious user.

```
function swapExactETHForTokens(...) external payable {
    address dTokenOut = dTokenOut_referrer[0];
    address payable referrer = address(uint160(dTokenOut_referrer[1]));
    require(dTokenOut != address(this),"cannot buy and sell same token");
    require(dTokenOut_referrer.length == 2 &&
comptroller.dTokenApproved(dTokenOut),"!dTokenOut");

    (uint256 mintiUSD, uint256 reserveTradeFee,) = amountsOut(address(this),
address(0), msg.value, msg.sender, referrer);
    iUSDbalance -= int(mintiUSD);

    if (referrer != address(0)) {
        doTransferOut(referrer,reserveTradeFee);
    } else {
        totalReserves += reserveTradeFee;
    }

    VTokenInterface(dTokenOut).sendTokenOut(mintiUSD, _minOut, _sendTo,
_deadline);
    require(iUSDrate() > int(-iUSDlimit),"sell would exceed iUSD limit.");

    emit SwapExactETHForTokens(dTokenOut, msg.value, mintiUSD);
}
```

Similar issues could happen in the other vToken implementations like the VBep20. If the pairing token is the underlying native currency token, then it may produce a reentrance vulnerability.

```
VTokenInterface(dTokenOut).sendTokenOut(valueUSD, _minOut, _sendTo,
_deadline);
```

Recommendation

The team is advised to move all the native currency transfers to the end of the method. That way the contract will guarantee that even if the recipient is a malicious user, all the contracts will have the proper state. As an extra security layer, the team could add some reentrancy-prevent modifiers.

PAO - Potential Arithmetic Overflow

Criticality	Critical
Status	Unresolved

Description

The contracts are using natively arithmetic operations. The ecosystem requires to be compiled in Solidity version lower than 8. As a result, the calculations are subject to integer overflows and underflows.

```
_amount * _price / 1e18;
...
rate = _iUSDrate * int(abs(_iUSDrate)) / 1e18;
...
```

Recommendation

The team is advised to use libraries that provide a set of functions for performing common arithmetic operations in a way that is resistant to overflows/underflows.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	VBep20.sol#L297
Status	Unresolved

Description

In the `swapExactTokensForTokens()` method of the vBep20 contract, the amount that is going to be transferred is calculated based on the provided amount (`_amountTokenIn`). The `doTransferIn()` method returns the actual amount that was transferred from the user to the contract. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

```
(uint256 valueUSD, uint256 reserveTradeFee,) = amountsOut(address(this),
address(0), _amountTokenIn, msg.sender, referrer);
```

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected. The contract could exploit the `doTransferIn()` method that returns the actual transferred amount.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts.

Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

`Actual Transferred Amount = Balance After Transfer - Balance Before Transfer`

FAD - Fixed Address Deployment

Criticality	Minor / Informative
Location	VTokenInterfaces.sol#L160
Status	Unresolved

Description

The contracts use internal variables to store the address of the contracts that they depend on. The variable `tradeModel` is fixed to the address `0xdC976ef337cce294bB7a09d9B1EeEc963c3942bc`. This assignment prevents the execution of the ecosystem in various environments. These environments should add a patch to reset the `tradeModel` address after deploy process. For instance, a unit test with mocked behavior cannot be implemented if the address points to a deployed address.

```
ITradeModel public tradeModel =
ITradeModel(0xdC976ef337cce294bB7a09d9B1EeEc963c3942bc);
```

Recommendation

The team is advised to keep the address state variables clear before the deployment. The contract could set these variables from the constructor. This way it will ease processes like the deployment in the testnet or the implementation of unit tests.

PHI - Permissions Handling Inconsistency

Criticality	Minor / Informative
Status	Unresolved

Description

The contract uses admin permissions in order to configure some variables that are essential for the proper operation. The code base contains two different ways of checking the admin permissions. The first one throws a descriptive error message about the failure. The second one has been implemented as a modifier and reverses the execution with a generic authorization message. The diversion of permission handling produced an inconsistency.

```
if (msg.sender != admin) {
    return fail(Error.UNAUTHORIZED,
FailureInfo.SET_PENDING_ADMIN_OWNER_CHECK);
}

modifier onlyAdmin() {
    require(msg.sender == admin,"!admin");
    _;
}
```

Recommendation

The team is advised to introduce one unique permission-handling mechanism. It is recommended to persist in the descriptive message pattern since it is more helpful for the users.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	TradeModel.sol#L15,29,41,42,43,44
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
bool isTradeModel = true
uint public referralDiscount = 0.10e18
uint public shrimpDiscount = 0.10e18
uint public fishDiscount    = 0.25e18
uint public sharkDiscount   = 0.50e18
uint public whaleDiscount   = 0.70e18
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	TradeModel.sol#L15 Comptroller.sol#L140,143
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
bool isTradeModel = true
uint internal constant closeFactorMinMantissa = 0.05e18
uint internal constant closeFactorMaxMantissa = 0.9e18
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	VToken.sol#L64,1432 TradeModel.sol#L82,93,104,117
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
limit = _limit;

tradeFeePerc = _tradeFeePerc
tradeReserveFactor = _tradeReserveFactor
shrimpThreshold = _shrimpThres
priceImpactLimit = _limit
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	TradeModel.sol#L230
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(newAvailableCash>=0,"remove liquidity exceed cash.")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	VToken.sol#L536,628,780,892 TradeModel.sol#L356 Comptroller.sol#L283,438
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
MintLocalVars memory vars
RedeemLocalVars memory vars
BorrowLocalVars memory vars
RepayBorrowLocalVars memory vars
uint _referralDiscount

;

err, , ui
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AggregatorV2V3Interface	Interface			
	latestAnswer	External		-
	latestTimestamp	External		-
	latestRound	External		-
	getAnswer	External		-
	getTimestamp	External		-
	decimals	External		-
	description	External		-
	version	External		-
	getRoundData	External		-
	latestRoundData	External		-
BEP20Interface	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

CarefulMath	Implementation			
	mulUInt	Internal		
	divUInt	Internal		
	subUInt	Internal		
	addUInt	Internal		
	addThenSubUInt	Internal		
CompDP	Implementation	ComptrollerV8Storage, ComptrollerInterfaceG2, ComptrollerErrorReporter, ExponentialNoError		
		Public	✓	-
	ensureAdmin	Private		
	ensureNonzeroAddress	Private		
	getAssetsIn	External		-
	checkMembership	External		-
	enterMarkets	External	✓	-
	addToMarketInternal	Internal	✓	
	exitMarket	External	✓	-
	mintAllowed	External	✓	onlyProtocolAllowed
	mintVerify	External	✓	-
	redeemAllowed	External	✓	onlyProtocolAllowed
	redeemAllowedInternal	Internal		
	redeemVerify	External	✓	-
	borrowAllowed	External	✓	onlyProtocolAllowed
	borrowVerify	External	✓	-

	repayBorrowAllowed	External	✓	onlyProtocolAllowed
	repayBorrowVerify	External	✓	-
	liquidateBorrowAllowed	External	✓	onlyProtocolAllowed
	liquidateBorrowVerify	External	✓	-
	seizeAllowed	External	✓	onlyProtocolAllowed
	seizeVerify	External	✓	-
	transferAllowed	External	✓	onlyProtocolAllowed
	transferVerify	External	✓	-
	getAccountLiquidity	Public		-
	getHypotheticalAccountLiquidity	Public		-
	getHypotheticalAccountLiquidityInternal	Internal		
	liquidateCalculateSeizeTokens	External		-
	liquidateVAICalculateSeizeTokens	External		-
	_setPriceOracle	External	✓	-
	_setCloseFactor	External	✓	-
	_setCollateralFactor	External	✓	-
	_setLiquidationIncentive	External	✓	-
	_setLiquidatorContract	External	✓	-
	_supportMarket	External	✓	-
	_addMarketInternal	Internal	✓	
	_setPauseGuardian	External	✓	-
	_setMarketBorrowCaps	External	✓	-
	_setBorrowCapGuardian	External	✓	-
	_setProtocolPaused	External	✓	validPauseState
	_setVAIController	External	✓	-
	_setVAIMintRate	External	✓	-
	_setTreasuryData	External	✓	-

	_become	External	✓	-
	adminOrInitializing	Internal		
	setVenusSpeedInternal	Internal	✓	
	_setComptrollerLens	External	✓	-
	updateVenusSupplyIndex	Internal	✓	
	updateVenusBorrowIndex	Internal	✓	
	distributeSupplierXDP	Internal	✓	
	distributeBorrowerXDP	Internal	✓	
	claimXDP	Public	✓	-
	claimXDP	Public	✓	-
	claimXDP	Public	✓	-
	claimXDP	Public	✓	-
	grantXDPInternal	Internal	✓	
	_grantXDP	External	✓	-
	_setVenusVAIVaultRate	External	✓	-
	_setVAIVaultInfo	External	✓	-
	_setXDPSpeed	External	✓	-
	getAllMarkets	Public		-
	getBlockNumber	Public		-
	setMintedVAIOf	External	✓	onlyProtocolAllowed
	releaseToVault	Public	✓	-
	getXDPAddress	Public		-
	_pauseTrading	External	✓	-
	dTokenApproved	External		onlyProtocolAllowed
ComptrollerInterfaceG1	Implementation			
	enterMarkets	External	✓	-
	exitMarket	External	✓	-

	mintAllowed	External	✓	-
	mintVerify	External	✓	-
	redeemAllowed	External	✓	-
	redeemVerify	External	✓	-
	borrowAllowed	External	✓	-
	borrowVerify	External	✓	-
	repayBorrowAllowed	External	✓	-
	repayBorrowVerify	External	✓	-
	liquidateBorrowAllowed	External	✓	-
	liquidateBorrowVerify	External	✓	-
	seizeAllowed	External	✓	-
	seizeVerify	External	✓	-
	transferAllowed	External	✓	-
	transferVerify	External	✓	-
	liquidateCalculateSeizeTokens	External		-
	setMintedVAIOf	External	✓	-
ComptrollerInterfaceG2	Implementation	ComptrollerInterfaceG1		
	liquidateVAICalculateSeizeTokens	External		-
ComptrollerInterface	Implementation	ComptrollerInterfaceG2		
	markets	External		-
	oracle	External		-
	getAccountLiquidity	External		-
	getAssetsIn	External		-
	claimVenus	External	✓	-
	venusAccrued	External		-
	venusSpeeds	External		-
	getAllMarkets	External		-

	venusSupplierIndex	External		-
	venusInitialIndex	External		-
	venusBorrowerIndex	External		-
	venusBorrowState	External		-
	venusSupplyState	External		-
	borrowCaps	Public		-
	getXDPAddress	Public		-
	dTokenApproved	External		-
IVAIVault	Interface			
	updatePendingRewards	External	✓	-
IComptroller	Interface			
	liquidationIncentiveMantissa	External		-
	treasuryAddress	External		-
	treasuryPercent	External		-
ComptrollerLens	Implementation	ComptrollerLensInterface, ComptrollerErrorReporter, ExponentialNoError		
	liquidateCalculateSeizeTokens	External		-
	liquidateVAICalculateSeizeTokens	External		-
	getHypotheticalAccountLiquidity	External		-
ComptrollerLensInterface	Interface			
	liquidateCalculateSeizeTokens	External		-
	liquidateVAICalculateSeizeTokens	External		-

	getHypotheticalAccountLiquidity	External		-
UnitrollerAdminStorage	Implementation			
ComptrollerV1Storage	Implementation	UnitrollerAdminStorage		
ComptrollerV2Storage	Implementation	ComptrollerV1Storage		
ComptrollerV3Storage	Implementation	ComptrollerV2Storage		
ComptrollerV4Storage	Implementation	ComptrollerV3Storage		
ComptrollerV5Storage	Implementation	ComptrollerV4Storage		
ComptrollerV6Storage	Implementation	ComptrollerV5Storage		
ComptrollerV7Storage	Implementation	ComptrollerV6Storage		
ComptrollerV8Storage	Implementation	ComptrollerV7Storage		
EIP20Interface	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-

	transfer	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	allowance	External		-
EIP20NonStandardInterface	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	allowance	External		-
ComptrollerErrorReporter	Implementation			
	fail	Internal	✓	
	failOpaque	Internal	✓	
TokenErrorReporter	Implementation			
	fail	Internal	✓	
	failOpaque	Internal	✓	
VAIControllerErrorReporter	Implementation			
	fail	Internal	✓	
	failOpaque	Internal	✓	
Exponential	Implementation	CarefulMath, ExponentialNoError		

	getExp	Internal		
	addExp	Internal		
	subExp	Internal		
	mulScalar	Internal		
	mulScalarTruncate	Internal		
	mulScalarTruncateAddUInt	Internal		
	divScalar	Internal		
	divScalarByExp	Internal		
	divScalarByExpTruncate	Internal		
	mulExp	Internal		
	mulExp	Internal		
	mulExp3	Internal		
	divExp	Internal		
ExponentialNoError	Implementation			
	truncate	Internal		
	mul_ScalarTruncate	Internal		
	mul_ScalarTruncateAddUInt	Internal		
	lessThanExp	Internal		
	lessThanOrEqualExp	Internal		
	greaterThanExp	Internal		
	isZeroExp	Internal		
	safe224	Internal		
	safe32	Internal		
	add_	Internal		
	add_	Internal		
	add_	Internal		
	add_	Internal		
	sub_	Internal		

	sub_	Internal		
	sub_	Internal		
	sub_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	mul_	Internal		
	div_	Internal		
	div_	Internal		
	div_	Internal		
	div_	Internal		
	div_	Internal		
	div_	Internal		
	div_	Internal		
	div_	Internal		
	fraction	Internal		
InterestRateModel	Implementation			
	getBorrowRate	External		-
	getSupplyRate	Public		-
ITradeModel	Interface			
	iUSDrate	External		-
	cashAddUSDMinusLoss	External		-
	newRemoveLiquidityAmt	External		-

	getCashAddUSDMultAbsRate	External		-
	amountsOut	External		-
JumpRateModel	Implementation	InterestRate Model		
		Public	✓	-
	utilizationRate	Public		-
	getBorrowRate	Public		-
	getSupplyRate	Public		-
LibNote	Implementation			
PriceOracle	Implementation			
	getUnderlyingPrice	External		-
SafeMath	Library			
	add	Internal		
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
TradeModel	Implementation	ITradeModel		
		Public	✓	-
	_setTradeFee	External	✓	onlyAdmin
	_setTradeReserveFactor	External	✓	onlyAdmin

	_updateTradeFeeDiscounts	External	✓	onlyAdmin
	setPriceImpactLimit	External	✓	onlyAdmin
	getValue	Public		-
	getAssetAmt	Public		-
	getAssetAmtInt	Public		-
	getValueInt	Public		-
	min	Public		-
	max	Public		-
	abs	Public		-
	iUSDrate	Public		-
	priceImpact	Public		-
	protocolLoss	Public		-
	removeLiquidityFee	Public		-
	newRemoveLiquidityAmt	Public		-
	adjustedPrice	Public		-
	cashAddUSDMinusLoss	Public		-
	getCashAddUSDMultAbsRate	External		-
	feeDiscount	Public		-
	amtAfterFee	Public		-
	amountOutUSDInternal	Public		-
	amountOutTokenInternal	Public		-
	amountsOut	External		-
Unitroller	Implementation	UnitrollerAdminStorage, ComptrollerErrorReporter		
		Public	✓	-
	_setPendingImplementation	Public	✓	-
	_acceptImplementation	Public	✓	-

	_setPendingAdmin	Public	✓	-
	_acceptAdmin	Public	✓	-
		External	Payable	-
VAI	Implementation	LibNote		
	rely	External	✓	note auth
	deny	External	✓	note auth
	add	Internal		
	sub	Internal		
		Public	✓	-
	transfer	External	✓	-
	transferFrom	Public	✓	-
	mint	External	✓	auth
	burn	External	✓	-
	approve	External	✓	-
	push	External	✓	-
	pull	External	✓	-
	move	External	✓	-
	permit	External	✓	-
VAIControllerInterface	Implementation			
	getVAIAddress	Public		-
	getMintableVAI	Public		-
	mintVAI	External	✓	-
	repayVAI	External	✓	-
	liquidateVAI	External	✓	-
	_initializeVenusVAIState	External	✓	-
	updateVenusVAIMintIndex	External	✓	-
	calcDistributeVAIMinterVenus	External	✓	-

VBep20	Implementation	VToken, VBep20Interface		
	initialize	Public	✓	-
	mint	External	✓	-
	redeemUnderlying	External	✓	-
	borrow	External	✓	-
	repayBorrow	External	✓	-
	repayBorrowBehalf	External	✓	-
	liquidateBorrow	External	✓	-
	getCashPrior	Internal		
	doTransferIn	Internal	✓	
	doTransferOut	Internal	✓	
	swapExactTokensForTokens	External	✓	-
VBep20Delegate	Implementation	VBep20, VDelegateInterface		
		Public	✓	-
	_becomeImplementation	Public	✓	onlyAdmin
	_resignImplementation	Public	✓	onlyAdmin
dTokenDelegator	Implementation	VTokenInterface, VBep20Interface, VDelegatorInterface		
		Public	✓	-
	_setImplementation	Public	✓	-
	mint	External	✓	-
	redeemUnderlying	External	✓	-
	borrow	External	✓	-

	repayBorrow	External	✓	-
	repayBorrowBehalf	External	✓	-
	liquidateBorrow	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	allowance	External		-
	balanceOf	External		-
	balanceOfUnderlying	External	✓	-
	getAccountSnapshot	External		-
	borrowRatePerBlock	External		-
	supplyRatePerBlock	External		-
	totalBorrowsCurrent	External	✓	-
	borrowBalanceCurrent	External	✓	-
	borrowBalanceStored	Public		-
	exchangeRateCurrent	Public	✓	-
	exchangeRateStored	Public		-
	getCash	External		-
	accrueInterest	Public	✓	-
	seize	External	✓	-
	_setPendingAdmin	External	✓	-
	_setComptroller	Public	✓	-
	_setReserveFactor	External	✓	-
	_acceptAdmin	External	✓	-
	_reduceReserves	External	✓	-
	_setInterestRateModel	Public	✓	-
	delegateTo	Internal	✓	
	delegateToImplementation	Public	✓	-
	delegateToViewImplementation	Public		-

		External	Payable	-
	_setLimitIUSD	External	✓	-
	_setTradeModel	External	✓	-
	getPriceToken	Public		-
	getExchangeCash	External		-
	iUSDrate	External		-
	removeAmountMinusFee	External		-
	getAvailableCash	External		-
	amountsOut	Public		-
	sendTokenOut	External	✓	-
	swapExactTokensForTokens	External	✓	-
dBTCB	Implementation	VBep20		
		Public	✓	-
dBNB	Implementation	VToken		
		Public	✓	-
	mint	External	Payable	-
	redeemUnderlying	External	✓	-
	borrow	External	✓	-
	repayBorrow	External	Payable	-
	repayBorrowBehalf	External	Payable	-
	liquidateBorrow	External	Payable	-
		External	Payable	-
	getCashPrior	Internal		
	doTransferIn	Internal	✓	
	doTransferOut	Internal	✓	
	requireNoError	Internal		
	swapExactETHForTokens	External	Payable	-

ChainlinkOracle	Implementation	PriceOracle		
		Public	✓	-
	setMaxStalePeriod	External	✓	onlyAdmin
	getUnderlyingPrice	Public		-
	getPrice	Internal		
	getChainlinkPrice	Public		-
	setUnderlyingPrice	External	✓	onlyAdmin
	setDirectPrice	External	✓	onlyAdmin
	setFeed	External	✓	onlyAdmin
	getFeed	Public		-
	assetPrices	External		-
	compareStrings	Internal		
	setAdmin	External	✓	onlyAdmin
VToken	Implementation	VTokenInterface, Exponential, TokenErrorReporter		
	initialize	Public	✓	-
	transferTokens	Internal	✓	
	transfer	External	✓	nonReentrant
	transferFrom	External	✓	nonReentrant
	approve	External	✓	-
	allowance	External		-
	balanceOf	External		-
	balanceOfUnderlying	External	✓	-
	getAccountSnapshot	External		-
	getBlockNumber	Internal		
	borrowRatePerBlock	External		-

	supplyRatePerBlock	External		-
	totalBorrowsCurrent	External	✓	nonReentrant
	borrowBalanceCurrent	External	✓	nonReentrant
	borrowBalanceStored	Public		-
	borrowBalanceStoredInternal	Internal		
	exchangeRateCurrent	Public	✓	nonReentrant
	exchangeRateStored	Public		-
	exchangeRateStoredInternal	Internal		
	getCash	External		-
	accrueInterest	Public	✓	-
	mintInternal	Internal	✓	nonReentrant
	mintFresh	Internal	✓	
	redeemUnderlyingInternal	Internal	✓	nonReentrant
	redeemFresh	Internal	✓	
	borrowInternal	Internal	✓	nonReentrant
	borrowFresh	Internal	✓	
	repayBorrowInternal	Internal	✓	nonReentrant
	repayBorrowBehalfInternal	Internal	✓	nonReentrant
	repayBorrowFresh	Internal	✓	
	liquidateBorrowInternal	Internal	✓	nonReentrant
	liquidateBorrowFresh	Internal	✓	
	seize	External	✓	nonReentrant
	seizeInternal	Internal	✓	
	_setPendingAdmin	External	✓	-
	_acceptAdmin	External	✓	-
	_setComptroller	Public	✓	-
	_setReserveFactor	External	✓	nonReentrant
	_setReserveFactorFresh	Internal	✓	
	_reduceReserves	External	✓	nonReentrant

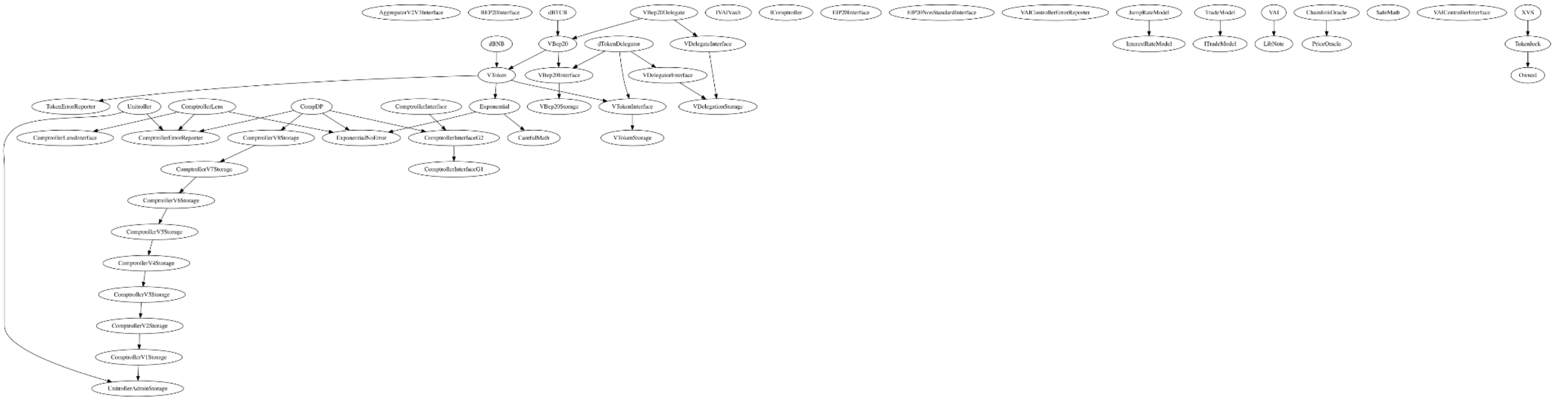
	_reduceReservesFresh	Internal	✓	
	_setInterestRateModel	Public	✓	-
	_setInterestRateModelFresh	Internal	✓	
	getCashPrior	Internal		
	doTransferIn	Internal	✓	
	doTransferOut	Internal	✓	
	iUSDrateLimits	Internal		
	_setLimitIUSD	External	✓	onlyAdmin
	_setTradeModel	External	✓	onlyAdmin
	getPriceToken	Public		-
	cashPlusUSD	Internal		
	getExchangeCash	Public		-
	iUSDrate	Public		-
	removeAmountMinusFee	Public		-
	getAvailableCash	Public		-
	amountsOut	Public		-
	sendTokenOut	External	✓	-
VTokenStorage	Implementation			
VTokenInterface	Implementation	VTokenStorage		
	transfer	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	allowance	External		-
	balanceOf	External		-
	balanceOfUnderlying	External	✓	-
	getAccountSnapshot	External		-
	borrowRatePerBlock	External		-

	supplyRatePerBlock	External		-
	totalBorrowsCurrent	External	✓	-
	borrowBalanceCurrent	External	✓	-
	borrowBalanceStored	Public		-
	exchangeRateCurrent	Public	✓	-
	exchangeRateStored	Public		-
	getCash	External		-
	accrueInterest	Public	✓	-
	seize	External	✓	-
	_setPendingAdmin	External	✓	-
	_acceptAdmin	External	✓	-
	_setComptroller	Public	✓	-
	_setReserveFactor	External	✓	-
	_reduceReserves	External	✓	-
	_setInterestRateModel	Public	✓	-
	getPriceToken	Public		-
	sendTokenOut	External	✓	-
	amountsOut	Public		-
VBep20Storage	Implementation			
VBep20Interface	Implementation	VBep20Storage		
	mint	External	✓	-
	redeemUnderlying	External	✓	-
	borrow	External	✓	-
	repayBorrow	External	✓	-
	repayBorrowBehalf	External	✓	-
	liquidateBorrow	External	✓	-
	swapExactTokensForTokens	External	✓	-

VDelegationStorage	Implementation			
VDelegatorInterface	Implementation	VDelegation Storage		
	_setImplementation	Public	✓	-
VDelegateInterface	Implementation	VDelegation Storage		
	_becomeImplementation	Public	✓	-
	_resignImplementation	Public	✓	-
Owned	Implementation			
		Public	✓	-
	transferOwnership	Public	✓	onlyOwner
Tokenlock	Implementation	Owned		
	freeze	Public	✓	onlyOwner
	unfreeze	Public	✓	onlyOwner
XVS	Implementation	Tokenlock		
		Public	✓	-
	allowance	External		-
	approve	External	✓	validLock
	balanceOf	External		-
	transfer	External	✓	validLock
	transferFrom	External	✓	validLock
	delegate	Public	✓	validLock
	delegateBySig	Public	✓	validLock
	getCurrentVotes	External		-

	getPriorVotes	Public		-
	_delegate	Internal	✓	
	_transferTokens	Internal	✓	
	_moveDelegates	Internal	✓	
	_writeCheckpoint	Internal	✓	
	safe32	Internal		
	safe96	Internal		
	add96	Internal		
	sub96	Internal		
	getChainId	Internal		

Inheritance Graph



Read the graphs with the original quality on

https://github.com/cyberscope-io/audits/blob/main/xdp

Flow Graph

Summary

Dual Pools contract implements a supply/borrow mechanism. This audit investigates security issues, business logic concerns and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors’ funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io