SMART CONTRACTS REVIEW

zokyo

November 30th 2023 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING CVI

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issue: 0 points deducted
- 0 High issues: 0 points deducted
- 3 Medium issues: 3 resolved = 0 points deducted
- 5 Low issues: = 6 resolved = 0 points deducted
- 7 Informational issues: 6 resolved and 1 acknowledged = -1 points deducted

TECHNICAL SUMMARY

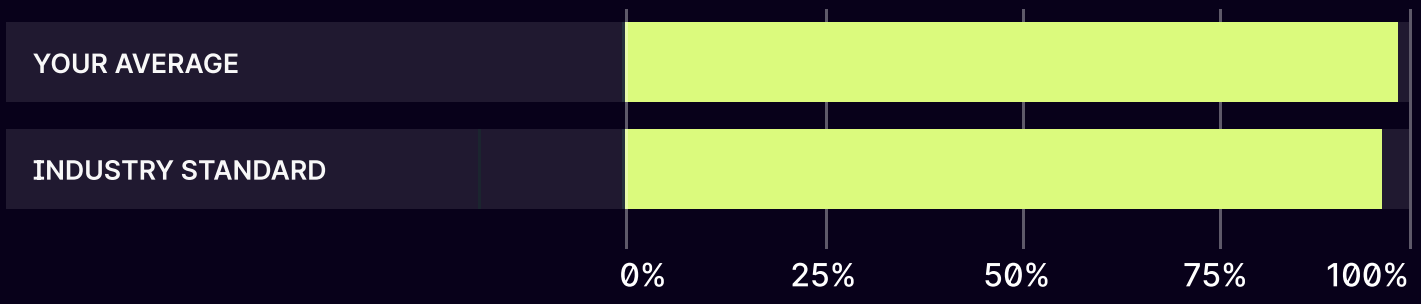
This document outlines the overall security of the CVI smart contracts evaluated by the Zokyo Security team.

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

Contract Status

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

Testable Code



95,44% 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 contracts 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 CVI team put in place a bug bounty program to encourage further active analysis of the smart contracts.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the CVI repository:
Repo: https://github.com/govi-dao/cvi-v4-contracts-release

Last commit - 7cfc2ed85baeb6428cb29b1b2f77fb1dcc02f3df

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

- CVIOracle.sol
- Platform.sol
- ThetaVaultRequestFulfillerV3.sol
- CVIReverseOracle.sol
- PlatformHelper.sol
- ThetaVaultV3.sol
- ETHVolOracle.sol
- PlatformMigrator.sol
- ThetaVaultV3Manager.sol
- FeesCalculator.sol
- PlatformRequestFulfillerV3.sol
- UCVIOracle.sol
- HedgedThetaVault.sol
- PositionRewards.sol
- UniswapHelper.sol
- KeepersBased.sol
- Rebaser.sol
- UniswapV3LiquidityManager.sol
- KeepersFeeVault.sol
- RebaserV3.sol
- VolatilityToken.sol
- Liquidation.sol
- ReferralManager.sol
- VolatilityTokenRequestFulfillerV3.sol
- LowLatencyRequestFulfiller.sol
- RequestFulfiller.sol
- VolatilityTokenV3.sol
- MegaThetaVault.sol
- ThetaVault.sol
- FeesCollector.sol
- Treasury.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 CVI 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 contracts logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team identified medium and low-severity vulnerabilities, along with a few informational issues. We would like to highlight that the CVI team promptly addressed and resolved all identified issues. For a more detailed breakdown of these findings, we recommend consulting 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” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the CVI team and the CVI team is aware of it, but they have chosen to not solve it. 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	Possible price manipulation due to dependency on manipulable spot price	Medium	Resolved
2	Inconsistent CVI value handling during the deposit in MegaThetaVault contract	Medium	Resolved
3	Missing Checks For Whether Arbitrum Sequencer Is Active	Medium	Resolved
4	Unvalidated referral codes allow users to claim undeserved affiliate benefits	Low	Resolved
5	Avoid using transfer for sending the native token	Low	Resolved
6	Missing _disableInitializer() implementation	Low	Resolved
7	Approve amount `0` for tokens to previously used routers/ managers	Low	Resolved
8	Method setPlatform does not check if the new platform is the same old platform	Low	Resolved
9	No address(0) check	Low	Resolved
10	Unsafe cast from int256 to uint256	Informational	Resolved
11	No need to initialize default values	Informational	Resolved
12	Use specific Solidity compiler version	Informational	Resolved
13	Unused import	Informational	Resolved
14	Lack of events in the implementation of privileged functions	Informational	Resolved
15	Poor NatSpec Comments	Informational	Acknowledged
16	Incorrect Comment Line	Informational	Resolved

Possible price manipulation due to dependency on manipulable spot price

The ThetaVault contract's _deposit function and the VolatilityToken contract's mintTokens function use a spot price as a fallback mechanism for determining the intrinsic token price when certain conditions are met (e.g., when balance or supply are zero). Specifically:

- In _deposit of ThetaVault, the spot price is fetched when the total supply of volToken is zero.
- In mintTokens of VolatilityToken, the spot price is fetched when both supply and balance are zero.

This spot price is sourced from a Uniswap V3 pool via the getSpotPrice function in the liquidity manager. If an attacker manipulates this price and then the _deposit or mintTokens function will be triggered under the conditions where the spot price is used (e.g., when supply or balance are zero), they might receive an unfair amount of tokens, or cause other users to receive a suboptimal amount, depending on the functionality.

Recommendation:
Use a TWAP to determine price, as this makes the attack more expensive and difficult to maintain a manipulated price.

Fix:
According to protocol's response:"Regarding using a spot price when the supply is zero: this doesn't really matter, as when balance is zero, the first mint gets just spot price times the initial amount of UDSC minted of vol tokens. As the first mint is the only owner of vol tokens, it doesn't matter what the sport price is and has no implication on the amount of minted tokens, as the next mint will be made proportionally to the now set supply of vol tokens anyway (i.e. if we just use a constant ratio, this would have been the same, except the ratio is set by spot price).

However, when the balance is `0`, this is different. In such a case it does matter, as there are volatility tokens existing in the supply. If that happens, it must mean that the volatility token's position was set to `0`, aka got liquidated. This scenario was not properly defended against so we added the following defense: if the volatility token position is liquidated, we do not use the spot price on the next mint, but use a fixed ratio, and we also restrict the mint to a certain max UDSC amount (say `100`). This is because this next mint will spread this new worth over all existing supply, causing the minter to lose a lot of value. However, as the amount is forced to be small, this is negligible. So, we both solve the liquidation of vol token position issue, and make sure we do not use the spot price in such a case. Following mints will be proportional as normal."

MEDIUM-2 | RESOLVED

Inconsistent CVI value handling during the deposit in MegaThetaVault contract

The depositForOwner function in the MegaThetaVault contract takes the _realTimeCVIValue as a parameter and assigns it to balanceCVIValue. The function then checks if _realTimeCVIValue is less than balanceCVIValue. Since they are initially the same, this condition will never be true, rendering this piece of logic useless.

```
  function depositForOwner(address _owner, uint168 _tokenAmount, uint32
_realTimeCVIValue) external override returns (uint256 thetaTokensMinted) {
      require(msg.sender == fulfiller);

      uint32 balanceCVIValue = _realTimeCVIValue;
      if (_realTimeCVIValue < balanceCVIValue) {
          balanceCVIValue = _realTimeCVIValue;
      }
```

As the implemented fulfill mechanism allows for operations delay, the mentioned function should retrieve the CVI value from an oracle, and then compare it with the passed _realTimeCVIValue and update the value accordingly.

Recommendation:
Retrieve the CVI value from the CVI oracle, compare it with _realTimeCVIValue and update the state accordingly.

MEDIUM-3 | RESOLVED

Missing Checks For Whether Arbitrum Sequencer Is Active

The protocol intends to deploy to L2. ChainLink recommends that users using price oracles, check whether the Arbitrum sequencer is active

https://docs.chain.link/data-feeds#l2-sequencer-uptime-feeds

If the sequencer goes down, oracles may have stale prices, since L2-submitted transactions (i.e. by the aggregating oracles) will not be processed.

Recommendation:
Use sequencer oracle inside the protocol, or else it might give stale prices for USDC and affect the liquidation process where we demand a certain peg from the returned amount.

LOW-1 | RESOLVED

Unvalidated referral codes allow users to claim undeserved affiliate benefits

In the _openPosition function within the Platform contract, openPositionFees is called with the user-provided referralCode. Inside the openPositionFees function, if _referralCode is not zero, it fetches the TierInfo and affiliate using the getInfoByCode function.
If a user provides a random referralCode that does not exist in the codeToAccount mapping of the referralManager, the codeToAccount mapping will return the default value for this mapping, which is address(0). As a result, when fetching the tierInfo corresponding to this non-existent affiliate, the accountToTier[address(0)] would be used. This will, in turn, again return a default value, which is 0. The effect of this is that tierToTierInfo[0] would be fetched, potentially giving the user the benefits of the TierInfo set at the constructor for the 0th tier.

Recommendation:
Add a validation check in the getInfoByCode function to ensure that the returned affiliate address is not address(0) before fetching the TierInfo. If it is address(0), revert the transaction or return a default TierInfo without any benefits.

LOW-2 | RESOLVED

Avoid using transfer for sending the native token

In Contract LowLatencyRequestFullFiller, the method refundExecutionFee(...) and method withdrawExecutionFee(...) use payable(...).transfer(...) for sending native token to an address. This uses a fixed 2300 gas and gas repricing may break this leading to execution fees not being refunded and being stuck in the contract forever resulting in loss of funds.

Recommendation:
Use .call() instead to transfer native tokens.

LOW-3 | RESOLVED

Missing _disableInitializer() implementation

The following contracts inherit the Intializable.sol and implement the initialize(...) method with the initializer modifier without disabling the initializers for the implementation contract as recommended by OpenZeppelin here.

MegaThetaVault.sol
HedgedThetaVault.sol
ThetaVault.sol
VolatilityToken.sol
Platform.sol
PlatfomHelper.sol
UniswapV3LiquidityManager.sol

Recommendation:
Disable the initializers for the implementation method as suggested by OpenZeppelin here.

LOW-4 | RESOLVED

Approve amount `0` for tokens to previously used routers/managers

In Contract HedgedThetaVault, the method setRewardRouter(...) approves HedhedThetaVault tokens of amount type(uint).max to _thetaRewardTracker but does not approve `0` to all previous _thetaRewardTracker addresses used. It can lead to malicious activity if left unchecked.

In Contract ThetaVault, the method setSwapRouter(...) approves token of amount type(uint).max to swapRouter but does not approve `0` to the old swap router address. The same method also approves vol tokens of amount type(uint).max to swapRouter without approving amount `0` to old swapRouter. The same applies to the method setLiquidityManager(...) as well. It can lead to malicious activity if left unchecked.

Recommendation:
It is advised to approve `0` to previously used addresses.

LOW-5 | RESOLVED

Method setPlatform does not check if the new platform is the same old platform

In Contract VolatilityToken, the method setPlatform(...) closes positions on the old platform and opens positions on the new platform. But it is not checking if the old platform is the same as the new platform which is advised to check.

Recommendation:
Add a check to ensure the new platform is not the same as the old platform.

LOW-6 | RESOLVED

No address(0) check

Contract HedgedThetaVault's constructor does not check if the parameter _inversePlatform is address(0). Once address(0) is set, there is no method to assign any other value.

Recommendation:
Add a check to ensure the _inversePlatform parameter is not address(0).

INFORMATIONAL-1 | RESOLVED

Unsafe cast from int256 to uint256

In the contract CVIOracle.sol, the method getTruncatedCVIVaule(...) casts int256 cviOracleValue to uint256 cviValue without checking if the CviVaule == cviOracleVaule.

Recommendation:
Add a check to ensure the cast is safe.

INFORMATIONAL-2 | RESOLVED

No need to initialize default values

In Contract Platform.sol, a few storage variables are being initialized to their default values which are not needed.

```
canPurgeLatestSnapshot = false;
emergencyWithdrawAllowed = false;
fulfiller = address(0);
isReverse = false;
isDepleted = false;
```

Recommendation:
Remove unnecessary initializations.

INFORMATIONAL-3 | RESOLVED

Use specific Solidity compiler version

Audited contracts use the following floating pragma:

```
pragma solidity ^0.8;
```

It allows to compile contracts with various versions of the compiler and introduces the risk of using a different version when deploying than during testing.

Recommendation:
Use a specific version of the Solidity compiler.

INFORMATIONAL-4 | RESOLVED

Unused import

In the Contract Elastic token, Initializable.sol is imported but not used.

Recommendation:
Remove unused imports.

INFORMATIONAL-5 | RESOLVED

Lack of events in the implementation of privileged functions

Some functions that carry out important changes to contracts state emit no events on changing significant parameters of the contracts by admin or other. This is taking place in a number of external state changing functions in the following files below.
In ThetaVault.sol:

- *setLiquidityManager*
- *setRange*
- *setSwapRouter*
- *setManager*
- *setRebaser*
- *setDepositor*
- *setDepositHoldings*
- *setMinPoolSkew*
- *setLiquidityPercentage*
- *setMinRebalanceDiff*
- *setInitialPrices*

In RequestFiller.sol

- *setRequestManager*
- *setEnableWhitelist*
- *setFulfillerAddress*

In MegaThetaVault.sol:

- *setFulfiller*
- *setDepositor*
- *setMinAmounts*

In ElasticToken.sol:

- *SetRebaser*

In CVIOracle.sol:

- *setDeviationCheck*
- *setMaxDeviation*

In FeesCalculator.sol:

- *setOracles*
- *setThetaVault*
- *setUCVIThetaVault*
- *setMaxAllowedUtilizationCollateral*
- *setFundingFeeMinRate*
- *setFundingFeeMaxRate*
- *setMinFundingFeeCviThreshold*
- *setMaxFundingFeeCviThreshold*
- *setFundingFeeDivisionFactor*
- *setFundingFeeCoefficients*
- *setFundingFeeMultiplier*
- *setStateUpdator*
- *setDepositFee*
- *setWithdrawFee*
- *setOpenPositionFee*
- *setClosePositionFee*
- *setOpenPositionLPFee*
- *setClosePositionLPFee*
- *setBuyingPremiumFeeMax*
- *setBuyingPremiumThreshold*
- *setCollateralToBuyingPremiumMapping*
- *setFundingFeeConstantRate*
- *setCollateralToExtraFundingFeeMapping*

In HedgedThetaVault.sol:

- *setFulfiller*
- *setRewardRouter*
- *setDepositHoldingsPercentage*
- *setWithdrawFeePercentage*

In KeepersBased.sol:

- *setUpkeepInterval*
- *setEnableWhitelist*
- *setOperatorAddress*

In KeepersFeeVault.sol:

- *setExtractor*

In Liquidation.sol:

- *setMinLiquidationThresholdPercents*
- *setMinLiquidationRewardPercent*
- *setMaxLiquidationRewardPercents*

In ThetaVaultManager.sol:

- *setPricePool*
- *setTwapInterval*
- *setMinHighSlippagePercentage*
- *setLowSlippageCVIWindow*
- *setHighSlippageCVIWindow*
- *setMaxDiviationPercentage*

Recommendation:

Emit relevant events to announce the changes.

INFORMATIONAL-6 | ACKNOWLEGED

Poor NatSpec Comments

Due to the nature of CVI protocol being complex, NatSpec comments would greatly influence the efficiency of outside reviewers reading the codebase. With proper NatSpec comments, a user might not interpret the intended functionality of the codebase.

Recommendation:
It is recommended to add NatSpec comments to ensure outside reviewers understand the intended functionality of the codebase.

INFORMATIONAL-7 | RESOLVED

Incorrect Comment Line

In the contract CVIReverseOracle, the function getComputedCVIValue has a comment line that states " // Truncate to 1 to avoid 0 cvi issues". While this is correct, it would be easier to understand as "rounding" to 1 instead of truncation.

Recommendation:
We recommend making the change to increase code readability and understanding of the codebase.

	CVIOracle.sol **Platform.sol** **ThetaVaultRequestFulfillerV3.sol** **CVIReverseOracle.sol** **PlatformHelper.sol** **ThetaVaultV3.sol** **ETHVolOracle.sol** **PlatformMigrator.sol**
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

	ThetaVaultV3Manager.sol FeesCalculator.sol PlatformRequestFulfillerV3.sol UCVIOracle.sol HedgedThetaVault.sol PositionRewards.sol UniswapHelper.sol KeepersBased.sol
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

	Rebaser.sol UniswapV3LiquidityManager.sol KeepersFeeVault.sol RebaserV3.sol VolatilityToken.sol Liquidation.sol ReferralManager.sol VolatilityTokenRequestFulfillerV3.sol
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

	LowLatencyRequestFulfiller.sol RequestFulfiller.sol VolatilityTokenV3.sol MegaThetaVault.sol ThetaVault.sol FeesCollector.sol Treasury.sol
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 Zokyo Security

As a part of our work assisting CVI in verifying the correctness of their contracts 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 CVI contracts requirements for details about issuance amounts and how the system handles these.

```
CVI Oracle Tests
    Gets CVI Round Data
    Gets Latest CVI Round Data
10000
    Gets Max Truncated CVI Value
    Sets deviation Check (109ms)
    Sets Max Deviation
    CVI Sanity Checks (38ms)
CVI For Inverse Oracle: 9000
CVI for Non-inverse Oracle: 1000
    Gets Truncated CVI Value From Reverse Oracle
66000
    Gets Max UCVI Truncated Value
When index is larger or equal to array length 66000
When index is shorter or equal to array length 6000
    Gets UCVI Truncated Value
66000
CVI For Inverse Oracle: 9000
CVI for Non-inverse Oracle: 0
    UCVI tests (171ms)
10000
    Reverse Oracle tests:  (149ms)
22000
   ETh Vol tests 1 (97ms)
CVI For Inverse Oracle: 9000
CVI for Non-inverse Oracle: 1000
    Eth vol tests 2 (82ms)
```

Fee Calculator Tests
  Sets Oracle (112ms)
  Sets theta vault (149ms)
  Sets UCVIThetaVault (160ms)
  Sets Max Utilization Collateral Ratio (239ms)
  Sets Funding Fee Min Rate (187ms)
  Sets Max Funding Fee Rate (222ms)
  Sets Min Funding Fee CVI Threshold (280ms)
  Sets Max Funding Fee CVI Threshold (279ms)
  Sets Funding Fee Division Factor (350ms)
  Sets Funding Fee Coeefecients (343ms)
  Sets Funding Fee Multiplier (355ms)
  Sets State Updator (544ms)
  Sets Deposit Fee (474ms)
  Sets Withdrawal Fee
  Sets Open Position Fees
  Sets Close Position Fees
  Sets Open Lp Position Fee
  Sets Close Lp Position Fee
  Sets Buying Premium Max Fee
  Sets Buying Premium Threshold
  Sets Collateral Buying Premium Fee Mapping (106ms)
  Sets funding Fee Constant Rate
  Sets Collateral To Extra Funding Fee Mapping (102ms)
100000000000
  Calculate Buying Premium Fee (797ms)
Collateral Ratio: 100000000000
Last Collateral Ratio 100000000000
  Calculate Buying Premium Fee Case 2: (57ms)
Collateral Ratio: 100000000000
Last Collateral Ratio 100000000000
  Calculate Buying Premium Fee Case 3: (59ms)
Collateral Ratio: 1000000000
Last Collateral Ratio 100000000000
  Calculate Buying Premium Fee Case 3: (111ms)
  Calculates close position fee (48ms)
Result: 6000000
Result2: 0
  calculates Single Unit Funding Fee: (141ms)
Result {
 '0': [
  '50000',

```
    '50000',
    discountPercentage: '50000',
    affiliatePercentage: '50000'
  ],
 '1': '0x70997970C51812dc3A010C7d01b50e0d17dc79C8',
  tierInfo: [
    '50000',
    '50000',
    discountPercentage: '50000',
    affiliatePercentage: '50000'
  ],
  affiliate: '0x70997970C51812dc3A010C7d01b50e0d17dc79C8'
}
[
  '50000',
  '50000',
  discountPercentage: '50000',
  affiliatePercentage: '50000'
]
     Tests Referral Manager Contract (111ms)
     Calculates open position fee (67ms)
     Calculates Single Unit Funding Fee
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,500,600,800,9
00,1100,1400,1600,2000,2400,2900,3500,4200,5200,6300,7700,9
400,11500,14000,17200,21200,26100,32300,39900,49500,61500,
76500,95300,100000,100000,100000,100000,100000,100000,100
000,100000,100000
     Gets Collateral To Buying Premium Mapping (53ms)
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
     Gets Collateral To Extra Funding Fee Mapping (51ms)
100000,114869,131950,151571,174110
     Gets Funding Coeffecients
1000000000000000000000
1701105871,0,1701106873,0,5000,false,false,false
    Updates Snapshot (50ms)
cviValue:  5000 periodEndRoundId:  2 periodEndTimestamp:
1701107098
10000000000,0,1701106889,2,10000,true,true,true
     Update Snapshot case 2:  (51ms)
```

Update Snapshot Case 3: (788ms)
cviValue: 5000 periodEndRoundId: 2 periodEndTimestamp: 1701107326
304879993906,304879993906,1701107225,2,10000,true,true,true
Update Snapshot Case 4: (80ms)
cviValue: 5000 periodEndRoundId: 2 periodEndTimestamp: 1701107461
304880018102,304880018102,1701107342,0,10000,true,false,true
Update Snapshot Case 4: end round id = start round id (58ms)
0
Calculate withdraw fee percent:
Result: 0
Result: 0
Calculates Single Unit Period Funding Fee
Calculates Single Unit Period Funding Fee Case 2: (44ms)
Calculates Single Unit Period Funding Fee Case 3: (904ms)
Result: 1000
Calculates Single Unit Period Funding Fee Case 4: (63ms)
Result: 306
Calculates Single Unit Period Funding Fee Case 5: (310ms)

Fees Collector
Can not be initialized again
Sets USDCETH Price Aggregator
Sets Router
Sets Staking Contract
Sets Staking Vault
Sets Arbitrum Contract
Sets Arbitrum Inbox
Sets Treasury
Sets Transfer Percentage
Sets Send Percentage
Sets Min Eth For Transfer
Sets Min USDC For Conversion
Tests Max Slippage
Sets To Send To Arbitrum
Sets Convert USDC
Sets Buy Back
Sets Funds Sender
Sets Enable Whitelist

```
    Sets Allowed Sender Address
    Sets New Native

Sets New Wrapped Token
Balance Of USDC In Contract Before Sending Profit: 0
Balance Of USDC In Contract After Sending Profit:
10000000000000000000
    Sends USDC Profit (43ms)
USDC Balance:  1000
WETH Balance:  2000
WETH Balance Of Treasury Before:  0
WETH Balance Of Treasury After:  60
    Tests Fund Transfer (203ms)
ETH Balance:  2000000000000000000
USDC Balance:  1000
ETH Balance Of Treasury Before:  2000000000000000000
ETH Balance Of Treasury After:  1600000000000000000
    Tests Fund Transfer When useNative Is True (1267ms)
ETH Balance:  2000000000000000000
USDC Balance:  1000
WETH Balance Of Staking Contract Before:  0
WETH Balance Of Staking Contract After:  340000000000000000
    Tests Funds Transfer When buyBack Is False (221ms)
ETH Balance:  2000000000000000000
USDC Balance:  1000
ETH Balance Of Contract Before:  2000000000000000000
ETH Balance Of Contract After:  0
    Tests Funds Transfer When sendToArbitrum Is True (227ms)
USDC Balance Before:  1000
USDC Balance After:  0
    Tests extraUSDC Transfer
USDC Balance:  1000
WETH Balance:  0
    Tests Not Enough Funds For Fund Transfer (81ms)
USDC Balance:  1000
WETH Balance:  1000
false
    Tests When sendToArbitrum And shouldConvert Is False
(107ms)
ETH Balance:  2000000000000000000
USDC Balance:  1000
false
```

Performs Upkeep (371ms)
ETH Balance: 2000000000000000000
USDC Balance: 1000
Same As Above But sendToArbitrum Is False (220ms)

HedgedThetaVault
initializes properly (53ms)
Checks If Fulfiller Can Only Be Set By The Owner (47ms)
Sets Deposit Holdings Percentage
Sets Withdraw Fee Percentage
Sets Reward Router
100000000000000000000000000
0
50000000000000000000
Tests Deposit (1217ms)
50000000000000000000
50
Tests Withdraw (2983ms)
19877938759
Adjusts Hedge Case 1 (1894ms)
true
Initializes Theta Vault Manager (1020ms)
Theta Vault Manager When Whitelist Is Disabled (378ms)

HedgedThetaVault
10000
Total Supply Before Deposit Is: 0
Total Balance After Deposit: 2000100
Total Supply After Deposit Is: 100000000000000
Tests deposit for owner branches (874ms)

Branches Tests
Total Supply 50000000000000000000
Balance: 200000000000000000000000000000000000
tokenHoldings 60000000000000000000
Inverse tokens: 60000000000000000000
HIII: 50000000
lpTokensToWithdraw: 10
AA: 0
AB: 0
AC: false
50000000000000000000 50

Tests Withdraw For Owner: (936ms)

Hedge test 3
OI Value: 0
totalInversePlatformBalance: 0
FINAL: 50002000000
Adjusts Hedge Case 3 (2082ms)

Hedge test 4
1) Adjusts Hedge Case 4

Hedge test 2
OI Value: 3000000000
totalInversePlatformBalance: 0
FINAL: 50002000000
Adjusts Hedge Case 2 (3882ms)
OI Value: 3000000000
totalInversePlatformBalance: 0
FINAL: 50002000000
Adjusts Hedge Case 2 (1716ms)

Total Balance Test
Initial total balance before deposits: 2000000
Total Balance Of Mega Vault After Deposit: 200
Total Supply Of Mega Vault After Deposit: 100000000000000
Total Supply Of Inverse Platform Before : 0
Total Supply Of Inverse Platform After : 100000000000000
AA: 100000000000000
Total Balance After Deposit In Inv Platform & Mega Is: 2000100
Γ£ö Tests Total Balance: (294ms)

MegaThetaVault
50000000
Balance of user: 50000000000000000000
Balance after deposit: 6
Deposits for owner (4814ms)
50000000000000000000
200
Deposit case where balance > 0 (390ms)
Balance of user: 50000000000000000000
Tests normal deposit (3390ms)
Calculates OI (105ms)

MegaThetavault2
balance: 50000000 cviBalance: 25000000 ucviBalance: 25000000
Rebalance test (3684ms)
balance: 50000000 cviBalance: 25000000 ucviBalance: 25000000
Rebalance test 2: (3686ms)
tests setters (971ms)

Referral Manager Tests
Tests Referral Manager Contract (855ms)

Tests for ThetaVaultRequestFulfiller
Can only be initialized once
Tests min deposit and withdraw amount
Tests deposit and withdraw emits (107ms)
Tests Low Latency Req Fulfiller Setters (498ms)
false
Performs Event Match Case 1
false
Performs Event Match Case 2
Performs Event Match Case 3
false
Performs Event Match Case 4
Tests Execute Deposit when catch errors is true:
Tests Execute Deposit when catch errors is false: (94ms)
Balance of user: 100000000000000
res 99999950000000
Tests Execute Withdraw when catch errors is false: (193ms)
Balance of user: 100000000000000
res 99999950000000
Tests Execute Withdraw when catch errors is true: (170ms)
Tests Execute Hedged Deposit when catch errors is false: (146ms)
Tests Execute Hedged Deposit when catch errors is true: (134ms)
Balance of user: 100000000000000
res 99999950000000
Tests Execute Hedged Withdraw when catch errors is false: (363ms)
Balance of user: 100000000000000
res 99999950000000

```
    Tests Execute Hedged Withdraw when catch errors is true:
(282ms)
    Execute Case 1:  (93ms)
    Execute Case 2:  (172ms)
    Execute Case 3:  (156ms)
    Execute Case 4:  (909ms)
    Execute Case 5:  (329ms)

TVRF when expiration period sec is lesser
    Execute Case 4:  (52ms)

Uniswap Helper
TickAtSqrtRatio -887272
TickAtSqrtRatio 879683
    gets tick at sqrt ratio (97ms)
1461446703485210103287273052203988822378723970342
    gets sqrt ratio at tick
6277101735386680
15930919111324522769 5
    gets spot price
A:  993722898264613320
B 61771017353866807638
    get twap price delta (47ms)
25
25
    get liquidity for amount0
792
792
    gets liquidity for amount 1
396
396
    gets amount 0 for liquidity
1
1
    Γ£ö gets amount1 for liquidity

UniV3LiqManager Tests
    Sets Operator (233ms)
0x150b7a02
    Gets onERC721Received
6394649534
1000045807692702017140832593928550412579642478334
```

```
    Sets Range (108ms)
Position Id Is:  2
This Should Be True: true

Position Id After Burn Is:  0
    Adds DEX Liquidity (585ms)
undefined
    Removes DEX Liquidity (278ms)
    Updates Pool Price When isVolTokenOnly Is True  (146ms)
    Updates Pool Price When isVolTokenOnly Is False  (649ms)
    Calculates DEX Liq USDC Amount
    2) Calculates DepositMintVolTokensUSDCAmount
0xa065f7322ef0efc01FdED6B5936068A8DE9Cc551
100000000000000000000
VDB:  100000000000000000000
100000000000000000000

Gets Vault DEX Tokens (206ms)
Balance Of  Contract Before Collecting Fee:
100000000000000000000
Balance Of Contract After Collecting Fee:  0
    Collects Fees (210ms)
    Calculates Arbitrage Amount (443ms)

Calcuates Arb Amount
When targetPrice > sqrtP:  44721358287
When targetPrice < sqrtP:  43
    Calculates Arbitrage Amount Case 3 (83ms)

Every test but with isVolToken0 as true
    Sets Operator (309ms)
0x150b7a02
    Gets onERC721Received
6394649534
1000045807692702017140832593928550412579642478334
    Sets Range (95ms)
Position Id Is:  2
This Should Be True: true
Position Id After Burn Is:  0
    Adds DEX Liquidity (722ms)
undefined
    Removes DEX Liquidity (93ms)
```

Updates Pool Price When isVolTokenOnly Is True (140ms)
Updates Pool Price When isVolTokenOnly Is False (137ms)
Calculates DEX Liq USDC Amount
MaxPriceSqrtX96: 1000049773623411026204330487172
MinPriceSqrtX96: 4295128739
Calculates DepositMintVolTokensUSDCAmount (65ms)
0x6F38937607d51e398A7317810966D6aF317407c4
10000000000000000000
VDB: 10000000000000000000
10000000000000000000
Gets Vault DEX Tokens (222ms)
Balance Of Contract Before Collecting Fee:
10000000000000000000
Balance Of Contract After Collecting Fee: 0
Collects Fees (63ms)

VolatilityToken

rebaseCVI
✓ rebaseCVI
rebaseCVI
✓ rebaseCVI
mintTokensForOwner
✓ mintTokensForOwner (107ms)
burnTokensForOwner()
✓ burnTokensForOwner
mintTokens
✓ mintTokens (71ms)
setPlatform
✓ setPlatform (110ms)
setFeesCollector()
✓ setFeesCollector (101ms)
setDeviationParameters()
✓ setDeviationParameters (48ms)
setThetaVault()
✓ setThetaVault (51ms)
✓ setPositionManager (43ms)
✓ setCappedRebase
✓ setCVIOracle (38ms)
✓ setFeesCalculator (52ms)
✓ setFeesCalculator
burnTokens()
✓ burnTokens
✓ burnTokens
✓ burnTokens (157ms)

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	%UNCOVERED LINES
KeepersFeeVault.sol	100	100	100	100	
Liquidation.sol	90	83.33	100	89	
PlatformMigrator.sol	100	92.86	100	100	
PlatformRequestFulfillerV3.sol	92	84	100	90	
RebaserV3.sol	100	100	100	100	
ThetaVault.sol	94	82.72	100	93	
VolatilityToken.sol	98.68	86.27	100	93.04	
VolatilityTokenRequestFulfiller V3.sol	90.5	80	100	88	
Platform.sol	91.38	79	100	89	
PlatformHelper.sol	94.60	83.63	100	93.4	
FeesCalculator.sol	91.7	94.17	100	97.99	
HedgedThetaVault.sol	96.2	83.87	100	95.41	
MegaThetaVault.sol	82.76	100	100	96.47	
CVIOracle.sol	100	100	100	100	
ThetaVaultRequestFulfiller V3.sol	90.54	87.5	100	91.57	
ReferralManager.sol	94.81	89	100	100	
CVIReverseOracle.sol	100	100	100	100	
ETHVolOracle.sol	100	100	100	100	
UCVIOracle.sol	100	100	100	100	
FeesCollector.sol	90.91	88.78	100	98	
ThetaVaultManager.sol	94.6	82.6	100	88	
LowLatencyRequestFulfiller.sol	100	80	100	100	

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	%UNCOVERED LINES
KeepersBased.sol	100	100	100	100	
UniswapHelper.sol	91.6	89.78	100	98.91	
UUniswapV3LiquidityManager.sol	97.81	85.33	100	100	
All Files	95.44	90.08	100	95.8	

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

zokyo