HAECHI AUDIT

Yeti Finance

Smart Contract Security Analysis

Published on : Dec 29, 2021

Version v2.1

HAECHI AUDIT

Smart Contract Audit Certificate

Yeti Finance

Security Report Published by HAECHI AUDIT
v1.0 Dec 17, 2021
v2.0 Dec 29, 2021

Auditor : Wook Yang

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	0	-	-	-	-
Minor	4	2	-	2	-
Tips	2	1	-	-	-

TABLE OF CONTENTS

4 Issues (0 Critical, 0 Major, 4 Minor) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring. We have the vision to *empower the next generation of finance*. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Universe,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io
Website: audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of the Yeti Finance smart contract created by Yeti team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Yeti team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends the Yeti team improve all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, Sample.sol:20 points to the 20th line of Sample.sol file, and Sample#fallback() means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found at GitHub (https://github.com/kingyetifinance/YetiFinance_Haechi_Audit/tree/master/packages/contracts). The last commit of the code used for this Audit is 6369c053b03121fa09a1280f9253032426f3c479.

Issues HAECHI AUDIT found 0 critical issues, 0 major issues, and 4 minor issues. There are 2 Tips issues explained that would improve the code's usability or efficiency upon modification.

Update [v.2.0] In the new commit *2dc3c0006aba32ba87f30d264f5da358dc5b47e5*, *c8102290cc98763743dc8614129cd84bca541104*, *e03c8a3c7a60ae9fbc27a0bb35994242971d6edb*, 2 minor issues, and 1 Tips issue have been revised.

Severity	Issue	Status
MINOR	The same collateral can be added as duplicates by using WhiteList#addCollateral().	(Found - v1.0) (Resolved - v2.0)
MINOR	The coll balance for an invalid amount can be updated.	(Found - v1.0) (Acknowledged - v2.0)
MINOR	There is an invalid require syntax.	(Found - v1.0) (Resolved - v2.0)
MINOR	Overflow may occur.	(Found - v1.0)
TIPS	An unused internal function exists.	(Found - v1.0) (Resolved - v2.0)
TIPS	An unused input parameter exists.	(Found - v1.0)
Notice	The privilege of the Owner is deleted in some cases.	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- ActivePool.sol
- BorrowerOperations.sol
- CollSurplusPool.sol
- DefaultPool.sol
- GasPool.sol
- HintHelpers.sol
- MultiTroveGetter.sol
- PriceFeed.sol
- SortedTroves.sol
- StabilityPool.sol
- TroveManager.sol
- TroveManagerLiquidations.sol
- TroveManagerRedemptions.sol
- YUSDToken.sol
- calculateMaxWithdrawHelper.sol
- leverUp.sol
- Dependencies
  - AggregatorV3Interface.sol
  - BaseMath.sol
  - CheckContract.sol
  - IERC20.sol
  - IERC2612.sol
  - ITellor.sol
  - LiquityBase.sol
  - LiquityMath.sol
  - LiquitySafeMath128.sol
  - Ownable.sol
  - SafeMath.sol
  - TellorCaller.sol
  - TroveManagerBase.sol
  - Whitelist.sol
  - YetiCustomBase.sol
- LPRewards
  - Pool2UniPool.sol
  - Unipool.sol
- PriceCurves

- ➢ ThreePieceWiseLinearPriceCurve.sol
- ❖ Proxy
  - ➢ BorrowerOperationsScript.sol
  - ➢ BorrowerWrappersScript.sol
  - ➢ ETHTransferScript.sol
  - ➢ SYETIScript.sol
  - ➢ StabilityPoolScript.sol
  - ➢ TokenScript.sol
  - ➢ TroveManagerScript.sol
- ❖ Swapper
  - ➢ Swapper.sol
- ❖ YETI
  - ➢ CommunityIssuance.sol
  - ➢ LockupContract.sol
  - ➢ LockupContractFactory.sol
  - ➢ ShortLockupContract.sol
  - ➢ YETIToken.sol
  - ➢ sYETIToken.sol

Yeti Finance smart contract has the following privileges.

- ❖ Owner

Each privilege can access the following functions.

Role	Functions
Owner	❖ ActivePool#setAddresses() ❖ BorrowerOperations#setAddresses() ❖ CollSurplusPool#setAddresses() ❖ DefaultPool#setAddresses() ❖ HintHelpers#setAddresses() ❖ PriceFeed#setAddresses() ❖ StabilityPool#setAddresses() ❖ SortedTroves#setParams() ❖ TroveManager#setAddresses() ❖ TroveManagerLiquidations#setAddresses() ❖ TroveManagerRedemptions#setAddresses() ❖ Whitelist#setAddresses() ❖ Whitelist#addCollateral()

- Whitelist#deprecateCollateral()
- Whitelist#undeprecateCollateral()
- Whitelist#changeOracle()
- Whitelist#changePriceCurve()
- Whitelist#changeRatio()
- Pool2Unipool#setParams()
- Pool2Unipool#setReward()
- Unipool#setParams()
- ThreePieceWiseLinearPriceCurve#adjustParams()
- ThreePieceWiseLinearPriceCurve#setAddresses()
- Swapper#setAddresses()
- LockupContractFactory#setYETITokenAddress()
- CommunityIssuance#setAddresses()
- ThreePieceWiseLinearPriceCurve#setAddresses()
- sYETIToken#setAddresses()
- sYETIToken#buyBack()
- sYETIToken#setTransferRatio()

Notice

The privilege of the Owner is deleted in some cases.

Most contracts that initialize by using the *setAddressess()* function change the privilege of the Ower to address(0) through *renounceOwnership()*. By doing so, the privilege of the Owner in the contract concerned is permanently deleted.
In the case of a contract where a function that contains the *onlyOwner* modifier no longer exists, it is unlikely to cause serious issues.

FINDINGS

MINOR

The same collateral can be added as duplicates by using WhiteList#addCollateral().

(Found - v.1.0)(Resolved - v.2.0)

```
function addCollateral(
    address _collateral,
    uint256 _minRatio,
    address _oracle,
    uint256 _decimals,
    address _priceCurve
) external onlyOwner {
    checkContract(_collateral);
    checkContract(_oracle);
    checkContract(_priceCurve);

    if (validCollateral.length != 0 && validCollateral[0] != _collateral) {
        require(collateralParams[_collateral].index == 0, "collateral already exists");
    }

    validCollateral.push(_collateral);
    collateralParams[_collateral] = CollateralParams(
        _minRatio,
        _oracle,
        _decimals,
        true,
        _priceCurve,
        validCollateral.length - 1
    );

    activePool.addCollateralType(_collateral);
    defaultPool.addCollateralType(_collateral);
    stabilityPool.addCollateralType(_collateral);
    collSurplusPool.addCollateralType(_collateral);

    emit CollateralAdded(_collateral);
}
```

Issue

Whitelist#addCollateral() has the statement that reverts a transaction when the same collateral is added. However, validCollateral[0], i.e., collateral that is added first, can be added as duplicates.

Example

For instance, if collateral A is added first and collateral B is added thereafter, adding collateral B as duplicate is reverted while adding collateral A as duplicate is not reverted.

Recommendation

If the above implementation is unintended, it is recommended to restrict all behaviors of adding duplicate collateral.

Update

[v2.0] - The issue has been resolved in a new commit *2dc3c0006aba32ba87f30d264f5da358dc5b47e5* by modifying the logic of if statement in *Whitelist#addCollateral()* function, so that collateral can no longer be duplicated.

MINOR

The coll balance for an invalid amount can be updated.

(Found - v.1.0)(Acknowledged - v.2.0)

```
function accountSurplus(
    address _account,
    address[] memory _tokens,
    uint256[] memory _amounts
) external override {
    _requireCallerIsTroveManager();
    balances[_account] = _sumColls(balances[_account], _tokens, _amounts);
    emit CollBalanceUpdated(_account);
}
```

Issue

CollSurpluPool#accountSurPlus() updates a user's balance information. A user must input the type and amount of collateral she intends to update. At this time, however, no statement confirms the validity of the function's input parameter.

Recommendation

We recommend adding a statement that confirms the validity of the input to prevent unnecessary calculation on invalid values and reduce the risk of potential problems.

Update

This issue has been acknowledged by Yeti finance. Also, this issue will not be valid if Yeti finance team does not change the troveManager address.

MINOR

There is an invalid require syntax.

(Found - v.1.0)(Resolved - v.2.0)

```
function openTrove(
    uint256 _maxFeePercentage,
    uint256 _YUSDAmount,
    address _upperHint,
    address _lowerHint,
    address[] memory _colls,
    uint256[] memory _amounts
) external override {
    _requireValidDepositCollateral(_colls);
    require(_amounts.length == _colls.length, "length");
    require(_amounts.length != 0, "Amounts == 0");
    LocalVariables_openTrove memory vars;
    bool isRecoveryMode = _checkRecoveryMode();
    ContractsCache memory contractsCache = ContractsCache(troveManager, activePool, yusdToken);
    _requireValidMaxFeePercentage(_maxFeePercentage, isRecoveryMode);
    _requireTroveisNotActive(contractsCache.troveManager, msg.sender);
    _requireNoOverlapCollsWithItself(_colls);
    vars.YUSDFee;
    vars.netDebt = _YUSDAmount;
    if (!isRecoveryMode) {
        vars.YUSDFee = _triggerBorrowingFee(
            contractsCache.troveManager,
            contractsCache.yusdToken,
            _YUSDAmount,
            _maxFeePercentage
        );
    }
    vars.VC = _getVC(_colls, _amounts);
    vars.YUSDFee = vars.YUSDFee.add(
        _getTotalVariableDepositFee(_colls, _amounts, vars.VC, 0, _YUSDAmount, contractsCache)
    );
    vars.netDebt = vars.netDebt.add(vars.YUSDFee);     _requireAtLeastMinNetDebt(vars.netDebt);
        vars.compositeDebt = _getCompositeDebt(vars.netDebt);
    assert(vars.compositeDebt > 0);

    vars.ICR = LiquityMath._computeCR(vars.VC, vars.compositeDebt);
    if (isRecoveryMode) {
        _requireICRisAboveCCR(vars.ICR);
    } else {
        _requireICRisAboveMCR(vars.ICR);
```

```
        uint256 newTCR = _getNewTCRFromTroveChange(vars.VC, true, vars.compositeDebt, true); // bools: coll increase, debt increase
        _requireNewTCRisAboveCCR(newTCR);
    }

    contractsCache.troveManager.setTroveStatus(msg.sender, 1);
    contractsCache.troveManager.updateTroveColl(msg.sender, _colls, _amounts);
    contractsCache.troveManager.increaseTroveDebt(msg.sender, vars.compositeDebt);
    contractsCache.troveManager.updateTroveRewardSnapshots(msg.sender);
    contractsCache.troveManager.updateStakeAndTotalStakes(msg.sender);
    sortedTroves.insert(msg.sender, vars.ICR, _upperHint, _lowerHint);
    vars.arrayIndex = contractsCache.troveManager.addTroveOwnerToArray(msg.sender);
    emit TroveCreated(msg.sender, vars.arrayIndex);

    require(
        _transferCollateralsIntoActivePool(msg.sender, _colls, _amounts),
        "BOps: Transfer collateral into ActivePool failed"
    );
    _withdrawYUSD(
        contractsCache.activePool,
        contractsCache.yusdToken,
        msg.sender,
        _YUSDAmount,
        vars.netDebt
    );
    _withdrawYUSD(
        contractsCache.activePool,
        contractsCache.yusdToken,
        gasPoolAddress,
        YUSD_GAS_COMPENSATION,
        YUSD_GAS_COMPENSATION
    );
    emit TroveUpdated(
        msg.sender,
        vars.compositeDebt,
        _colls,
        _amounts,
        BorrowerOperation.openTrove
    );
    emit YUSDBorrowingFeePaid(msg.sender, vars.YUSDFee);
}
```

Issue

BorrowerOperations#openTrove() function allows users to generate a trove and locks the collateral in the contract. The user must input the type and amount of collateral via the input parameter. At this time, according to the error message, the function must confirm that the collateral amount is not 0.
However, under the current implementation, the function confirms whether *amounts.length* is 0, not whether each of the *amounts[]* values is 0.

Recommendation

We recommend adding a require statement that confirms whether each of the *amounts[]* values is 0 in line with the error message.

Update

[v2.0] - The problem has been resolved in a new commit *c8102290cc98763743dc8614129cd84bca541104* by adding require syntax to check if each of *amounts[]* has a value of 0.

MINOR

Overflow may occur.

(Found - v.1.0)(Acknowledged - v.2.0)

Issue

Functions implemented in the *withdrawHelper* contract are confirmed not to use safeMath in the process of calculating the token amount, including the balance.

Recommendation

To prevent potential overflow, we recommend using safeMath for calculation.

Update

This issue has been acknowledged by Yeti finance. Also, this issue will not be valid if Yeti finance team does not change the frontend.

TIPS

An unused internal function exists.

(Found - v.1.0)(Resolved - v.2.0)

Issue

Some internal functions of the *StabilityPool* contract are implemented yet are not in use. Such functions are *_requireCallerIsActivePool()*, *_requireCallerIsTroveManager()*, and *_requireUserHasTrove()*.

Recommendation

We advise deleting the unused internal functions.

Update

[v2.0] - The issue has been resolved in a new commit *e03c8a3c7a60ae9fbc27a0bb35994242971d6edb* by removing unused internal functions.

TIPS

An unused input parameter exists.

(Found - v.1.0)(Acknowledged - v.2.0)

```
function setParams(
    address _yetiTokenAddress,
    address _uniTokenAddress,
    uint _duration
)
    external
    override
    onlyOwner
{
    checkContract(_yetiTokenAddress);
    checkContract(_uniTokenAddress);

    uniToken = IERC20(_uniTokenAddress);
    yetiToken = IYETIToken(_yetiTokenAddress);

    emit YETITokenAddressChanged(_yetiTokenAddress);
    emit UniTokenAddressChanged(_uniTokenAddress);
}
```

Issue

Pool2Unipool contract initializes a contract by using the *setParam()* function. At this time, _duration among the input parameters of the setParam function is not being used by the function.

Recommendation

We advise deleting the unused parameter, _duration.

Update

This issue has been acknowledged by Yeti finance.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects in Avalanche. To write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results are unit test results that cover the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to having issues.

```
ActivePool
  setAddresses()
    ✓ should set appropriate contract (47ms)
  sendCollaterals()
    ✓ should fail when invalid input
    valid case
      ✓ should transfer collateral tokens
  increaseYUSDDebt()
    valid case
      ✓ should emit ActivePoolYUSDDebtUpdated event
      ✓ should increase yusd debt
  decreaseYUSDDebt()
    valid case
      ✓ should emit ActivePoolYUSDDebtUpdated event
  getCollateral()
    ✓ should return appropriate balance
  getAllCollateral()
    ✓ should return appropriate token
    ✓ should return appropriate balance
  getCollateralVC()
    ✓ should return appropriate balance
  getVC()
    ✓ should return appropriate balance
BorrowerOperations
  setAddresses()
    ✓ should set appropriate contract
  openTrove()
    ✓ should fail when invalid input (62ms)
    ✓ should fail when invalid input amount
    valid case
      ✓ should transfer collateral tokens to activePool contract
      ✓ should insert hint
      ✓ should emit TroveCreated event
      ✓ should emit YUSDBorrowingFeePaid event
  addColl()
    valid case
```

```
    ✓ should transfer collateral out tokens to activePool contract
    ✓ should reInsert hint
    ✓ should emit YUSDBorrowingFeePaid event
    ✓ should emit YUSDBorrowingFeePaid event
  withdrawColl()
    valid case
    ✓ should transfer collateral out tokens to activePool contract
    ✓ should reInsert hint
    ✓ should emit YUSDBorrowingFeePaid event
    ✓ should emit YUSDBorrowingFeePaid event
  withdrawYUSD()
    valid case
    ✓ should transfer collateral out tokens to activePool contract
    ✓ should reInsert hint
    ✓ should emit YUSDBorrowingFeePaid event
    ✓ should emit YUSDBorrowingFeePaid event
  repayYUSD()
    valid case
    ✓ should transfer collateral out tokens to activePool contract
    ✓ should reInsert hint
    ✓ should emit YUSDBorrowingFeePaid event
    ✓ should emit YUSDBorrowingFeePaid event
  adjustTrove()
    valid case
    ✓ should transfer collateral out tokens to activePool contract
    ✓ should reInsert hint
    ✓ should emit YUSDBorrowingFeePaid event (567ms)
    ✓ should emit YUSDBorrowingFeePaid event
  closeTrove()
    ✓ should fail when insufficient yusd balance (303ms)
    valid case
    ✓ should transfer collateral out tokens to activePool contract
    ✓ should repay yusd with activePool
    ✓ should burn yusd
  claimCollateral()
    valid case
    ✓ should decrease balances
    ✓ should transfer collateral
  getTotalVariableDepositFee()
    ✓ should mint yusdToken to sYeti contract (1094ms)
withdrawHelper
  calculateCollateralVC()
    ✓ should fail when invalid input
    valid case
    ✓ should calc appropriate value VC (73ms)
```

```
  calculateCollateralVCFee()
    ✓ should fail when invalid input
    valid case
      ✓ should calc appropriate fee (55ms)
      ✓ should calc appropriate value (62ms)
CollSurPlusPool
  setAddresses()
    ✓ should set appropriate contract
  accountSurplus()
    valid case
      ✓ should increase balances
      ✓ should emit CollBalanceUpdated event
  claimColl()
    ✓ should fail when not add coll yet
    valid case
      ✓ should decrease balances
      ✓ should emit CollateralSent event
      ✓ should transfer collateral
  getCollateral()
    ✓ should return appropriate balance
  getAllCollateral()
    ✓ should return appropriate token
    ✓ should return appropriate balance
  getAmountClaimable()
    ✓ should return 0 when account has not collateral balance
  getCollVC()
    ✓ should return appropriate balance (39ms)
DefaultPool
  setAddresses()
    ✓ should set appropriate contract
  sendCollsToActivePool()
    ✓ should fail when invalid input
    valid case
      ✓ should transfer collateral tokens
  increaseYUSDDebt()
    valid case
      ✓ should emit ActivePoolYUSDDebtUpdated event
      ✓ should increase yusd debt
  decreaseYUSDDebt()
    valid case
      ✓ should emit ActivePoolYUSDDebtUpdated event
  getCollateral()
    ✓ should return appropriate balance
  getAllCollateral()
    ✓ should return appropriate token
```

```
    ✓ should return appropriate balance
  getVC()
    ✓ should return appropriate balance (39ms)
HintHelpers
  setAddresses()
    ✓ should set appropriate contract
  getRedemptionHints()
    ✓ should return appropriate hint address (75ms)
    ✓ should return appropriate hint ICR (40ms)
    ✓ should return appropriate hint ICR (39ms)
  getApproxHint()
    ✓ should return appropriate hint address
    ✓ should return appropriate diff
    ✓ should return appropriate latestRandomSeed (40ms)
LockupContract
  withdrawYETI()
    ✓ should fail when not unlock yet
    ✓ should fail when msg.sender is not beneficiary
    ✓ should transfer yeti to beneficiery
Pool2Unipool
  setParams()
    ✓ should set params
  setReward()
    ✓ should set reward
  stake()
    ✓ should update pool info
    ✓ should transfer token to pool
  withdraw()
    ✓ should transfer token to user
  withdrawAndClaim()
    ✓ should transfer yeti to user
PriceCurve
  getFee()
    ✓ should return 1e18 when max fee
    ✓ should return appropriate fee
  getFeeAndUpdate()
    ✓ should return 1e18 when max fee
  getFeePoint()
    ✓ should return appropriage collateral fee(1)
    ✓ should return appropriage collateral fee(2)
    ✓ should return appropriage collateral fee(3)
SortedTroves
  setParams()
    ✓ should set appropriate parameters
  insert()
```

```
    valid case
      ✓ should insert data fisrt trial
      ✓ should insert data behind (53ms)
      ✓ should insert data front (57ms)
      ✓ should insert data between (102ms)
      ✓ should insert data between && invalid (83ms)
  remove()
    valid case
      ✓ should remove data when 1 data exist (49ms)
      ✓ should remove data behind (94ms)
      ✓ should remove data front (94ms)
      ✓ should remove data between (97ms)
  reInsert()
    valid case
      ✓ should reinsert data when 1 data exist (62ms)
  findInsertPosition()
    valid case
      ✓ should find valid insert position(1) (55ms)
      ✓ should find valid insert position(2) (58ms)
      ✓ should find valid insert position(3) (58ms)
      ✓ should find valid insert position(4)
  validInsertPosition()
    valid case
      ✓ should find valid insert position (52ms)
StabilityPool
  setAddresses()
    ✓ should set appropriate contract
  registerFrontEnd()
    ✓ should fail when invalid input (100ms)
    valid case
      ✓ should register frontend
  provideToSP()
    ✓ should fail when frotend not registered
    ✓ should fail when msg.sender already registered
    ✓ should fail when invalid amount input
    valid case
      ✓ should set frontend tag
      ✓ should payout yeti(1)
      ✓ should payout yeti(2)
      ✓ should transfer yusd to stabilityPool
  withdrawFromSP()
    ✓ should fail when msg.sender not deposited (117ms)
    valid case
      ✓ should payout yeti
      ✓ should payout yeti
```

```
      ✓ should transfer yusd to depositor
      ✓ should transfer collateral token to depositor
  offset()
    ✓ should fail when invalid input(1) (171ms)
    ✓ should fail when invalid input(2) (184ms)
    ✓ should return when initial trial
    valid case
      ✓ should decrease yusd debt
      ✓ should burn yusd
    valid case(2)
      ✓ should decrease yusd debt
      ✓ should burn yusd
  updateDepositAndSnapshots()
    ✓ should update snapshots
  getCollateral()
    ✓ should return appropriate balance
  getAllCollateral()
    ✓ should return appropriate token
    ✓ should return appropriate balance
  getVC()
    ✓ should return appropriate balance (41ms)
TellorCaller
  getTellorCurrentValue()
    ✓ should return appropriate retrive, value, timestamp(1)
    ✓ should return appropriate retrive, value, timestamp(2)
UniPool
  setParams()
    ✓ should set params
  stake()
    ✓ should update pool info
    ✓ should transfer token to pool
  withdraw()
    ✓ should transfer token to user
  withdrawAndClaim()
    ✓ should transfer yeti to user
WhiteList
  setAddresses()
    ✓ should set appropriate contract
  addCollateral()
    ✓ should fail when add same collateral
    valid case
      ✓ should add collaterals
      ✓ should update collateral information
  deprecateCollateral()
    ✓ should fail when collateral already depricated
```

```
  valid case
    ✓ should deprecate collaterals
undeprecateCollateral()
  ✓ should fail when collateral already activate
  valid case
    ✓ should undeprecate collaterals
changePriceCurve()
  ✓ should fail when collateral is not exist
  valid case
    ✓ should change oracle
changeOracle()
  ✓ should fail when collateral is not exist
  valid case
    ✓ should change oracle
changeRatio()
  ✓ should fail when collateral is not exist
  ✓ should fail when too large ratio
  valid case
    ✓ should change oracle
getFee()
  ✓ should return appropriate fee percentage
getFeeAndUpdate()
  ✓ should update appropriate fee
getValueUSD()
  ✓ should return appropriate amount
```

End of Document