HAECHI AUDIT

Krome

Smart Contract Security Analysis

Published on : Mar 28, 2022

Version v3.0

HAECHI AUDIT

Smart Contract Audit Certificate

Krome

Security Report Published by HAECHI AUDIT
v1.0 Mar 07, 2022
v2.0 Mar 11, 2022
v3.0 Mar 28, 2022

Auditor : Felix Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	1	1	-	-	All Issues Resolved
Minor	-	-	-	-	-
Tips	-	-	-	-	-

TABLE OF CONTENTS

0 Issues (0 Critical, 1 Major, 0 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 smart contract created by Krome team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Krome 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 Krome 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/krome-finance/krome-contract). The last commit of the code used for this Audit is “3e39d8df9e1fe5ec7ad8ace886f892549e93e4e5”.

Issues	HAECHI AUDIT found 0 critical issues, 1 major issue, and 0 minor issues. There are 0 Tips issues explained that would improve the code’s usability or efficiency upon modification.
Update	One major issue has been resolved from the new commit, “b15e4b3e2f59b57737dddf6247a82290c314a753”,

Severity	Issue	Status
⚠ **MAJOR**	The UsdkPriceOracle#getLatestPrice() function may return inappropriate values.	(Found - v1.0) (Resolved -v.2.0)

OVERVIEW

Contracts subject to audit

- FXS1559_AMO_V3
- IAMO
- Context
- Owned
- ReentrancyGuard
- TimelockOwned
- ERC20
- ERC20Custom
- ERC20CustomMock
- ERC20KIP7
- ERC20Mock
- ERC20MockTWAMM
- ERC721Receiver
- ERC721ReceiverMock
- IERC20_Detailed
- IERC20
- IKIP7Receiver
- SafeERC20
- IUniswapV2Factory
- IUniswapV2Pair
- IUniswapV2Router02
- Timelock
- ClaimswapHelper
- ITokenSwapHelper
- IPresale
- PresaleByKlaytn
- PresaleVesting
- IKrome
- KromeShares
- TokenVesting
- Address
- BytesLib
- TransferHelper
- Math
- SafeMath

- AMOMock
- CollateralRatioMock
- ExecutorMock
- SwapMock
- TokenSwapHelperMock
- UniswapPricePairMock
- IPairPriceOracle
- IPriceOracle
- IPair
- KromePriceOracle
- MockPriceOracle
- MockStaticPairPriceOracle
- UsdkPriceOracle
- TreasuryERC20_Mock
- TreasuryUsdkUniswapPair
- GaugeMock
- GaugeRewardsDistributor
- IGaugeController
- IGaugeRewardsDistributor
- IKromeMiddlemanGauge
- IStakingBoostController
- IStakingTreasury
- VotingEscrow
- KromeGaugeController
- IFarm
- IVeKrome
- KromeRewardForStakingDelegator
- ManualGaugeController
- StakingBoostController
- IERC20Decimals
- StakingRewardComptroller
- StakingRewardsMultiGauge
- IRewardComptroller
- StakingTreasury_ERC20
- IAMOMinter
- ICollatBalance
- IUsdk
- IUsdkPool
- ICollateralRatioProvider
- KromeStablecoin
- UsdkAMOMinter

- UsdkCollateralRatio
- UsdkPoolMock
- UsdkPoolV3
- VeDelegation
- DelegationProxy
- IDelegationProxy
- VotingEscrowDelegation
- VotingEscrowDelegationMock
- IveKrome
- IYieldDistributor
- SmartWalletWhitelist
- SmartWalletChecker
- IVotingEscrowDelegation
- veKrome
- veKromeYieldDistributorV4

FINDINGS

⚠ **MAJOR**

The UsdkPriceOracle#getLatestPrice() function may return inappropriate values. (Found - v.1.0) (Resolved - v.2.0)

```
    // price0 = reserve1/reserve0  3:1 = 3
    // in USD e9
    function getLatestPrice() external view override returns (uint256) {
        // UQ112×112 → E9
        uint256 usdkToKlay = (isUsdk0ForKlay ? pair_usdk_klay.price0CumulativeLast() :
pair_usdk_klay.price1CumulativeLast()) * PRICE_PRECISION / Q112;
        uint256 usdPerKlay = oracle_klay_usd.getLatestPrice();
        return usdkToKlay * usdPerKlay / PRICE_PRECISION;
    }
```

[https://github.com/krome-finance/krome-contract/Oracle/UsdkPriceOracle.sol#L45-L52]

Issue

In the *UsdkPriceOracle#getLatestPrice()* function, there can be cases where the decimal of usdPerKlay is not 9. If so, the *UsdkPriceOracle#getLatestPrice()* function may return an abnormally high or small value.

Recommendation

We recommend modifying the return statement of the *UsdkPriceOracle#getLatestPrice()* function to take decimal into account similar to the KromePriceOracle#getLatestPrice() function.

Update

[v2.0] - This issue has been resolved as the *UsdkPriceOracle#getLatestPrice()* function has been modified to take decimal into account.

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 on the mainnet. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering 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 existing issues.

```
FXS1559_AMO_V3
  FXS1559_AMO_V3 spec
   constructor
    ✔ set owner properly
    ✔ set USDK properly
    ✔ set KROME properly
    ✔ set yieldDistributor properly
    ✔ set amo_minter_address properly
    ✔ set amo_minter properly
    ✔ set pool properly
    ✔ set tokenSwapHelper properly
    ✔ set max_slippage properly
    ✔ set burn_fraction properly
    ✔ set custodian_address properly
    ✔ set timelock_address properly
   #dollarBalances
    ✔ should return dollar balances properly
   #swapBurn
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ swap USDK and burn KROME tokens
      ✔ notify reward if it exist
   #burnUsdk
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ approve and burn USDK tokens
   #burnKrome
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ approve and burn KROME tokens
   #setBurnFraction
    ✔ should fail if _burn_fraction is too high
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ set _burn_fraction properly
    #setUsdkPool
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set pool properly
    #setAMOMinter
      ✔ should fail if timelock is zero address
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set amo_minter,custodian and timelock properly
    #setSafetyParams
      ✔ should fail if max_slippage is too high
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set max_slippage properly
    #setTokenSwapHelper
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set tokenSwapHelper properly
    #setYieldDistributor
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set yieldDistributor properly
    #recoverERC20
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ transfer token to msg.sender

UsdkPoolV3 and UsdkAMOMinter
  UsdkPoolV3 spec
    constructor
      ✔ set USDK properly
      ✔ set KROME properly
      ✔ set timelock address properly
      ✔ set custodian address properly
      ✔ set priceFeedUsdkUsd properly
      ✔ set priceFeedKromeUsd properly
      ✔ set collateral_addresses properly
      ✔ set collateral information properly
      ✔ set pool_ceilings properly
      ✔ set oracle_usdk_usd_decimals properly
      ✔ set oracle_krome_usd_decimals properly
    #collateral_information
      ✔ should fail if collat_address is invalid
      valid case
    ✔ should return collateral_information properly
  #allCollaterals
   ✔ should return collateral_addresses properly
  #getUsdkPrice
   ✔ should return USDK price properly
  #getKromePrice
   ✔ should return KROME price properly
  #getUsdkInCollateral
   ✔ should return USDK value in collateral tokens properly
  #freeCollatBalance
   ✔ should return collat balance properly
  #collatDollarBalance
   ✔ should return total balance properly
  #buybackAvailableCollat
   ✔ should return available value properly
  #recollatTheoColAvailableE18
   ✔ should return missing amount of collateral
  #recollatAvailableKrome
   ✔ should return available amount of KROME
  #curEpochHr
   ✔ should return current epoch hour
  #mintUsdk
   ✔ should fail if collateral is not enabled
   ✔ should fail if minting is paused
   ✔ should fail if USDK price is too low
   ✔ should fail if amount which will be minted lower than out_min
   ✔ should fail if try to mint over pool ceiling
   valid case(Collateral)
    ✔ transfer collateral tokens to contract
    ✔ mint USDK tokens to msg.sender
   valid case(Algorithmic)
    ✔ burn krome tokens
    ✔ mint USDK tokens to msg.sender
   valid case(Fractional)
    ✔ burn krome tokens
    ✔ transfer collateral tokens to contract
    ✔ mint USDK tokens to msg.sender
  #estimateMint
   ✔ should fail if USDK price is too high
   valid case
    ✔ should return USDK,collat and KROME value properly
  #redeemUsdk
   ✔ should fail if collateral is not enabled
   ✔ should fail if redeem is paused
```

```
    ✔ should fail if USDK price is too high
    ✔ should fail if try to redeem over contract balances
    ✔ should fail if collat_out is lower than col_out_min
    ✔ should fail if krome_out is lower than krome_out_min
    valid case(Collateral)
      ✔ increase redeemCollateralBalances
      ✔ increase unclaimedPoolCollateral
      ✔ update lastRedeemed of msg.sender
      ✔ burn USDK tokens
    valid case(Algorithmic)
      ✔ increase redeemKROMEBalances
      ✔ increase unclaimedPoolKROME
      ✔ update lastRedeemed of msg.sender
      ✔ burn USDK tokens
      ✔ mint KROME tokens
    valid case(Fractional)
      ✔ increase redeemCollateralBalances
      ✔ increase unclaimedPoolCollateral
      ✔ increase redeemKROMEBalances
      ✔ increase unclaimedPoolKROME
      ✔ update lastRedeemed of msg.sender
      ✔ burn USDK tokens
      ✔ mint KROME tokens
  #estimateRedeem
    ✔ should fail if USDK price is too high
    valid case
      ✔ should return collat and KROEM value properly
  #collectRedemption
    ✔ should fail if redeem is paused
    ✔ should fail if try to collect too soon
    valid case(Collateral)
      ✔ set redeemCollateralBalances and unclaimedPoolCollateral properly
      ✔ should transfer collateral to msg.sender
    valid case(Algorithmic)
      ✔ set redeemKROMEBalances and unclaimedPoolKROME properly
      ✔ should transfer KROME to msg.sender
    valid case(Fractional)
      ✔ set redeemCollateralBalances and unclaimedPoolCollateral properly
      ✔ should transfer collateral to msg.sender
      ✔ set redeemKROMEBalances and unclaimedPoolKROME properly
      ✔ should transfer KROME to msg.sender
  #buyBackKrome
    ✔ should fail if collateral is not enabled
    ✔ should fail if buyBack is paused
    ✔ should fail if collateral balances is not positive
    ✔ should fail if collateral balances is insufficient for buyBack
    ✔ should fail if col_out is lower than col_out_min
    valid case
      ✔ burn KROME token
      ✔ transfer collateral token to msg.sender
      ✔ set bbkHourlyCum properly
  #toggleMRBR
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ tog_idx = 0 - should change mintPaused and emit MRBRToggled event
      ✔ tog_idx = 1 - should change redeemPaused and emit MRBRToggled event
      ✔ tog_idx = 2 - should change buyBackPaused and emit MRBRToggled event
      ✔ tog_idx = 3 - should change recollateralizePaused and emit MRBRToggled event
  #addAMOMinter
    ✔ should fail if msg.sender is not authorized
    ✔ should fail if try to add zero address
    ✔ should fail if try to add invalid AMO
    valid case
      ✔ add AMO minter and emit AMOMinterAdded event
  #removeAMOMinter
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ remove AMO minter and emit AMOMinterRemoved event
  #setCollateralPrice
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ set collateral_prices and emit CollateralPriceSet event
  #toggleCollateral
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ set enabled_collaterals and emit CollateralToggled event
  #setPoolCeiling
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ set pool_ceilings and emit PoolCeilingSet event
  #setFees
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ set fee informatio and emit FeesSet event
  #setPoolParameters
    ✔ should fail if msg.sender is not authorized
    valid case
      ✔ set bonus_rate, redemption_delay and emit PoolParametersSet event
    #setPriceThresholds
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set mint_price_threshold, redeem_price_threshold and emit PriceThresholdsSet event
    #setBbkRctPerHour
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set bbkMaxColE18OutPerHour, rctMaxKromeOutPerHour and emit BbkRctPerHourSet
event
    #setOracles
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set oracle info of KROME and USDK and emit OraclesSet event
    #setCustodian
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set custodian address and emit CustodianSet event
    #setTimelock
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set timelock address and emit TimelockSet event
  UsdkAMOMinter spec
    constructor
      ✔ set USDK properly
      ✔ set KROME properly
      ✔ set timelock address properly
      ✔ set custodian address properly
      ✔ set pool addresses properly
      ✔ set collateral address properly
      ✔ set pool_ceilings properly
      ✔ set missing_decimals properly
    #collatDollarBalance
      ✔ should return collat balance properly
    #dollarBalances
      ✔ should return usdk_val_e18 and collat_val_e18
    #allAMOAddresses
      ✔ should return all AMO address
    #allAMOsLength
      ✔ should return the number of AMOs
    #usdkTrackedGlobal
      ✔ should return tracked USDK
    #usdkTrackedAMO
      ✔ should return tracked USDK of specific AMO
    #syncDollarBalances
  ✔ update stored dollar balances
#mintUsdkForAMO
  ✔ should fail if msg.sender is not authorized
  ✔ should fail if amo address is invalid
  ✔ should fail if try to mint over mint cap
  ✔ should fail if CR would be too low
  valid case
    ✔ mint USDK to AMO
    ✔ update stored dollar balances
#burnUsdkFromAMO
  ✔ should fail if msg.sender is invalid AMO
  valid case
    ✔ burn USDK
    ✔ update minted balances
    ✔ update stored dollar balances
#mintKromeForAMO
  ✔ should fail if msg.sender is not authorized
  ✔ should fail if amo address is invalid
  ✔ should fail if try to mint over mint cap
  valid case
    ✔ mint KROME to AMO
    ✔ update stored dollar balances
#burnKromeFromAMO
  ✔ should fail if msg.sender is invalid AMO
  valid case
    ✔ burn KROME
    ✔ update minted balances
    ✔ update stored dollar balances
#giveCollatToAMO
  ✔ should fail if msg.sender is not authorized
  ✔ should fail if amo address is invalid
  ✔ should fail if try to give over borrow cap
  valid case
    ✔ borrow collateral from pool
    ✔ transfer collateral to AMO
    ✔ update borrowed balances
    ✔ update stored dollar balances
#receiveCollatFromAMO
  ✔ should fail if msg.sender is invalid AMO
  valid case
    ✔ transfer collateral to pool
    ✔ update borrowed balances
    ✔ update stored dollar balances
#addAMO
```

```
  ✔ should fail if msg.sender is not authorized
  ✔ should fail if try to add zero address
  ✔ should fail if try to add invalid AMO address
  ✔ should fail if try to add already added AMO address
  valid case
    ✔ add AMO address and update balances and offsets
    ✔ update stored dollar balances if needed
    ✔ should emit AMOAdded event
#removeAMO
  ✔ should fail if msg.sender is not authorized
  ✔ should fail if try to remove zero address
  ✔ should fail if try to remove already removed AMO address
  valid case
    ✔ remove AMO address
    ✔ update stored dollar balances if needed
    ✔ should emit AMORemoved event
#setCustodian
  ✔ should fail if msg.sender is not authorized
  ✔ should fail if try to set zero address
  valid case
    ✔ set custodian address properly
#setUsdkMintCap
  ✔ should fail if msg.sender is not authorized
  valid case
    ✔ set USDK mint cap properly
#setKromeMintCap
  ✔ should fail if msg.sender is not authorized
  valid case
    ✔ set KROME mint cap properly
#setCollatBorrowCap
  ✔ should fail if msg.sender is not authorized
  valid case
    ✔ set borrow cap properly
#setMinimumCollateralRatio
  ✔ should fail if msg.sender is not authorized
  valid case
    ✔ set min_cr properly
#setAMOCorrectionOffsets
  ✔ should fail if msg.sender is not authorized
  valid case
    ✔ set offsets properly
    ✔ update stored dollar balances
#setUsdkPool
  ✔ should fail if msg.sender is not authorized
```

```
      ✔ should fail if collateral mismatches
      valid case
        ✔ set pool address properly
    #recoverERC20
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ transfer tokens to owner

KromePriceOracle
  KromePriceOracle spec
    constructor
      ✔ set KROME address properly
      ✔ set oracle_klay_usd properly
      ✔ set pair_usdk_klay properly
      ✔ set pair_usdk_krome properly
      ✔ set klay_price_precision properly
      ✔ set isUsdk0ForKlay properly
      ✔ set isUsdk0ForKrome properly
    #getDecimals
      ✔ should return number 9
    #getLatestPrice
      ✔ should return latest price properly
    #consult
      ✔ should return token price properly

KromeShares
  KromeShares spec
    constructor
      ✔ should fail if _timelock_address is zero address
      valid case
        ✔ set name properly
        ✔ set symbol properly
        ✔ set owner properly
        ✔ set timelock address properly
        ✔ mint KROME token properly
    #setUsdkAddress
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if try to set USDK zero address
      valid case
        ✔ set USDK address and emit UsdkAddressSet event
    #pool_burn_from
      ✔ should fail if msg.sender is not valid USDK pools
      valid case
        ✔ burn KROME tokens and emit KromeBurned event
    #pool_mint
      ✔ should fail if msg.sender is not valid USDK pools
      valid case
        ✔ mint KROME tokens and emit KromeMinted event
    #toggleVotes
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ change trackingVotes state
    #transfer
      ✔ track votes if trackingVotes is true
    #transferFrom
      ✔ track votes if trackingVotes is true
    #getCurrentVotes
      ✔ should return current votes of address
    #getPriorVotes
      ✔ should return prior votes of address

KromeStablecoin
  KromeStablecoin spec
    constructor
      ✔ should fail if _timelock_address is zero address
      valid case
        ✔ set name properly
        ✔ set symbol properly
        ✔ set creator properly
        ✔ mint usdk token properly
    #global_collateral_ratio
      ✔ should return collateral ratio properly
    #oracle_price
      ✔ should fail if choice is invalid
      valid case
        ✔ choice == 0 should return USDK price
        ✔ choice == 1 should return KROME price
    #usdk_price
      ✔ should return USDK price properly
    #krome_price
      ✔ should return KROME price properly
    #eth_usd_price
      ✔ should return eth_usd_price properly
    #usdk_info
      ✔ should return information of USDK properly
    #usdk_pools_length
      ✔ should return number of USDK pools
    #globalCollateralValue
      ✔ should return total collateral value properly
    #pool_burn_from
      ✔ should fail if msg.sender is not valid USDK pools
      valid case
        ✔ burn USDK tokens and emit UsdkBurned event
    #pool_mint
      ✔ should fail if msg.sender is not valid USDK pools
      valid case
        ✔ mint USDK tokens and emit UsdkMinted event
    #addPool
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if try to add zero address
      ✔ should fail if try to add already added address
      ✔ should fail if try to add invalid USDK pool
      valid case
        ✔ add USDK pool and emit PoolAdded event
    #removePool
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if try to remove zero address
      ✔ should fail if try to remove already removed address
      valid case
        ✔ remove USDK pool and emit PoolRemoved event
    #setKromeAddress
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if set Krome zero address
      valid case
        ✔ set Krome address and emit KromeAddressSet event
    #setEthUsdOracle
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if set eth_usd_pricer zero address
      valid case
        ✔ set eth_usd_pricer, eth_usd_pricer_decimals and emit EthUsdOracleSet event
    #setController
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if set controller zero address
      valid case
        ✔ set controller address and emit ControllerSet event
    #setCollateralRatioProvider
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if set collateral ratio provider zero address
      valid case
        ✔ set collateral ratio provider address and emit CollateralRatioProviderSet event
    #setUsdkEthOracle
      ✔ should fail if msg.sender is not authorized
```

```
      ✔ should fail if set usdk_oracle zero address
      valid case
        ✔ set usdk_eth_oracle address and emit UsdkEthOracleSet event
    #setKromeEthOracle
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if set krome_oracle zero address
      valid case
        ✔ set krome_eth_oracle address and emit KromeEthOracleSet event

UsdkCollateralRatio
  UsdkCollateralRatio spec
    constructor
      ✔ should fail if _timelock_address is zero address
      valid case
        ✔ set USDK properly
        ✔ set collateral_ratio properly
        ✔ set frax_step properly
        ✔ set refresh_cooldown properly
        ✔ set price_target properly
        ✔ set price_band properly
    #refreshCollateralRatio
      ✔ should fail if ratio is paused
      ✔ should fail if refresh cooldown is not passed
      valid case
        case 1 - current price is high
          ✔ decrease collateral ratio and emit CollateralRatioRefreshed event
        case 2 - current price is low
          ✔ increase collateral ratio and emit CollateralRatioRefreshed event
    #setFraxStep
      ✔ should fail if msg.sender is not authorized
      ✔ should fail if try to set step too high value
      valid case
        ✔ set frax_step and emit FraxStepSet event
    #setPriceTarget
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set price_target and emit PriceTargetSet event
    #setRefreshCooldown
      ✔ should fail if msg.sender is not authorized
      valid case
        ✔ set refresh_cooldown and emit RefreshCooldownSet event
    #setPriceBand
      ✔ should fail if msg.sender is not authorized
      valid case
```

```
          ✔ set price_band and emit PriceBandSet event
        #setController
          ✔ should fail if msg.sender is not authorized
          ✔ should fail if try to set controller zero address
          valid case
            ✔ set controller_address and emit ControllerSet event
        #toggleCollateralRatio
          ✔ should fail if msg.sender is not authorized
          valid case
            ✔ change collateral_ratio_paused and emit CollateralRatioToggled event

    UsdkPriceOracle
      UsdkPriceOracle spec
        constructor
          ✔ set USDK address properly
          ✔ set oracle_klay_usd properly
          ✔ set pair_usdk_klay properly
          ✔ set klay_price_precision properly
          ✔ set isUsdk0ForKlay properly
        #getDecimals
          ✔ should return number 9 (42ms)
        #getLatestPrice
          1) should return latest price properly
        #consult
          ✔ should return token price properly

  GaugeRewardsDistributor
      construction
        ✔ curator address set properly
        ✔ reward token address set properly
        ✔ gauge controller address set properly
        ✔ distributionsOn flag set true
      #currentReward
        ✔ at first, current reward should return 0
        ✔ Over time, it returns a reward as much as the gauge ratio each contract has.
      #distributeReward
        ✔ should fail if distribute flag is off (44ms)
        ✔ should fail if gauge does not registered in whitelist
        valid case
          ✔ before a one gauge duration passed there is no reward (51ms)
          ✔ when one gauge duration passed, distributor starts to distribute rewards (549ms)
          ✔ As time goes by, the amount of rewards received increases (1647ms)
          ✔ should emit RewardsDistributed event (1637ms)
      #recoverERC20
    ✔ should fail if msg.sender is not owner/gov
    valid case
      ✔ owner get tokens
      ✔ should emit RecoveredERC20 event
  #setGaugeState
    ✔ should fail if msg.sender is not owner/gov
    ✔ should fail if gauge address is ZERO_ADDRESS
    ✔ should fail if gauge address is not added
    valid case
      ✔ update is_middleman state
      ✔ update gauge_whitelist state
      ✔ should emit GaugeStateChanged event
  #setGaugeDuration
    ✔ should fail if msg.sender is not owner/gov
    ✔ should fail if gauge address is ZERO_ADDRESS
    ✔ should fail if gauge address is not registered
    valid case
      ✔ update gauge_duration
  #addGauge
    ✔ should fail if msg.sender is not owner/gov (44ms)
    ✔ should fail if try to add ZERO_ADDRESS
    ✔ should fail if try to add already registered gauge contract
    ✔ should fail 0 duration given (41ms)
    valid case
      ✔ update gauge_whitelist
      ✔ update gauge_period_finish_time
      ✔ update gauge_duration
  #setCurator
    ✔ should fail if msg.sender is not owner/gov
    ✔ update curator address (valid)
  #setGaugeController
    ✔ should fail if msg.sender is not owner/gov
    ✔ update curator address (valid)

KromeGaugeController
  construction
    ✔ token set properly
    ✔ voting escrow set properly
    ✔ contract deployer set to the owner
  #add_gauge
    ✔ should fail if msg.sender is not owner
    ✔ should fail if invalid gauge types
    ✔ should fail if try to add the same gauge
    valid case - weight == 0
```

```
      ✔ number of gauges update
      ✔ target address gauge type set properly
      ✔ should emit NewGauge event
    valid case - weight > 0
      ✔ number of gauges update
      ✔ target address gauge type set properly
      ✔ total weight should update
      ✔ should emit NewGauge event
  #add_type
    ✔ should fail if msg.sender is not owner
    valid case
      ✔ update n_gauge_types
      ✔ update gauge_type_names
      ✔ should emit AddType event
  #change_global_emission_rate
    ✔ should fail if msg.sender is not owner
    valid case
      ✔ update global_emission_rate
      ✔ should emit GlobalEmissionRate
  #change_type_weight
    ✔ should fail if msg.sender is not owner
    valid case
      ✔ update total weight
      ✔ should emit NewTypeWeight event
  #change_gauge_weight
    ✔ should fail if msg.sender is not owner
    valid case
      ✔ update total weight
      ✔ should emit NewGaugeWeight event
    #vote_for_gauge_weights
      ✔ should fail if token expires before next_time
      ✔ should fail if invalid voting power parameter
      ✔ should fail if try to vote non-gauge address
      valid case - full vote
        ✔ update user voted power (38ms)
        ✔ should emit VoteForGauge event
      valid case - partial vote
        ✔ update user voted power
        ✔ should emit VoteForGauge event
      valid case - inc/dec vote
        ✔ update user voted power (63ms)
        ✔ should emit VoteForGauge event (63ms)
      valid case - set to 0
        ✔ update user voted power (256ms)
        ✔ should emit VoteForGauge event (231ms)

  KromeRewardForStakingDelegator
    #collectRewardForWithdraw
      ✔ should fail if invalid farm given
      valid case - nothing to claim
        ✔ there is no collected rewards
      valid case - something to claim
        ✔ collected reward information update (1732ms)
    #collectRewardForVeKrome
      ✔ should fail if invalid farm given
      ✔ should fail if vekrome_lock_time is less than minimum (41ms)
      valid case - collectReward == 0
        ✔ there is no collected reward
      valid case - collectReward > 0
        ✔ user veKrome balance increase (6199ms)
        ✔ no withdraw information exist (5913ms)
    #withdrawLockedWithRewardLock
      ✔ should fail if invalid farm given
      ✔ should fail if staking is not found
     valid case
        ✔ collect reward information update (1034ms)
    #withdrawLockedVeKromeLock
      ✔ should fail if invalid farm given
      ✔ should fail if staking is not found
     valid case
        ✔ collect reward information update (1034ms)
   #setter functions
       ✔ should fail if msg.sender does not have appropriate role
      valid case
        ✔ set contract state properly
        ✔ should emit event (if exist)

 staking / farming scenario tests (multiple contracts involves)
    ✔ sanity check after environment setup
    basic testing
      ✔ scenario 1
      ✔ scenario 2
      ✔ scenario 3
      ✔ scenario 4
      ✔ scenario 5
      ✔ scenario 6
      ✔ scenario 7
      ✔ scenario 8
    ✔ scenario 9
    ✔ scenario 10
    ✔ scenario 11
    ✔ scenario 12
    ✔ scenario 13
    ✔ scenario 14
   random seed testing (each test runs multiple times due to random feature)
     ✔ scenario 1
     ✔ scenario 2
     ✔ scenario 3
     ✔ scenario 4
     ✔ scenario 5
     ✔ scenario 6
     ✔ scenario 7
     ✔ scenario 8
     ✔ scenario 9
     ✔ scenario 10
     ✔ scenario 11
     ✔ scenario 12
     ✔ scenario 13
     ✔ scenario 14

 veKROME
   #commit_smart_wallet_checker
     ✔ should fail if msg.sender is not owner
     valid case
       ✔ set future_smart_wallet_checker address
       ✔ should emit SmartWalletCheckerComitted event
   #appoly_smart_wallet_checker
     ✔ should fail if msg.sender is not owner
     valid case
       ✔ set smart_wallet_checker address
       ✔ should emit SmartWalletCheckerApplied event
   #toggleEmergencyUnlock
     ✔ should fail if msg.sender is not owner
     valid case
       ✔ flip emergencyUnlockActive flag
       ✔ should emit EmergencyUnlockToggled event
   #toggleDepositDelegatorWhitelist
     ✔ should fail if msg.sender is not owner
     valid case
       ✔ set deposit_delegator_whitelist flag
       ✔ should emit DepositDelegatorWhitelistToggled event
   #recoverERC20
```

```
  ✔ should fail if msg.sender is not owner
  valid case
    ✔ owner get token (69ms)
#checkpoint
  ✔ record global data to checkpoint (716ms)
#create_lock
  ✔ should fail if _value == 0
  ✔ should fail if already lock exist (335ms)
  ✔ should fail if unlock time is before than now
  ✔ should fail if unlock time is later than 4 years (MAXTIME)
  valid case
    ✔ create lock
#deposit_for
  ✔ should fail if _value == 0
  ✔ should fail if there is no existing lock
  ✔ should fail if try to deposit for expired lock
  valid case
    ✔ lock information update (409ms)
increase_amount
  ✔ should fail if _value == 0
  ✔ should fail if there is no existing lock
  ✔ should fail if try to deposit for expired lock
  valid case
    ✔ increase lock amount (without modify the unlock time) (342ms)
#increase_unlock_time
  ✔ should fail if try to change unlock time of expired lock
  ✔ should fail if try to decrease lock duration
  ✔ should fail if try to increase more than 4 years
  valid case
    ✔ increase unlock time (303ms)
#manage_deposit_for
  case1 : _locked.amount == 0 (create_lock)
    ✔ should fail if try to change unlock time of expired lock
    ✔ should fail if try to decrease lock duration
    ✔ should fail if try to increase more than 4 years
    ✔ create_lock (320ms)
  case2 : _locked.amount > 0 (update lock)
    ✔ should fail if _value == 0
    ✔ should fail if try to deposit for expired lock
    valid case
      ✔ increase amount (344ms)
#withdraw
  ✔ should fail if try to withdraw non-expired lock
  valid case - case 1: normal withdraw
    ✔ withdraw success (419ms)
  valid case - case 2: emergency withdraw
    ✔ emergency withdraw success (193ms)
```

End of Document