

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

[TABLE OF CONTENTS](#)

[ABOUT US](#)

[INTRODUCTION](#)

[SUMMARY](#)

[OVERVIEW](#)

[FINDINGS](#)

[UsdkPriceOracle#getLatestPrice\(\) 함수가 적절하지 않은 값을 반환할 수 있습니다. \(Found - v.1.0\) \(Resolved - v.2.0\)](#)

[DISCLAIMER](#)

[Appendix A. Test Results](#)

ABOUT US

HAECHI AUDIT은 디지털 자산이 가져올 금융 혁신을 믿습니다. 디지털 자산을 쉽고 안전하게 만들기 위해 HAECHI AUDIT은 '보안'과 '신뢰'라는 가치를 제공합니다. 그로써 모든 사람이 디지털 자산을 부담없이 활용할 수 있는 세상을 꿈꿉니다.

HAECHI AUDIT은 글로벌 블록체인 업계를 선도하는 HAECHI LABS의 대표 서비스 중 하나로, 스마트 컨트랙트 보안 감사 및 개발을 전문적으로 제공합니다.

다년간 블록체인 기술 연구 개발 경험을 보유하고 있는 전문가들로 구성되어 있으며, 그 전문성을 인정받아 블록체인 기술 기업으로는 유일하게 삼성전자 스타트업 육성 프로그램에 선정된 바 있습니다. 또한, 이더리움 재단과 이더리움 커뮤니티 펀드로부터 기술 장려금을 수여받기도 하였습니다.

대표적인 클라이언트 및 파트너사로는 카카오 자회사인 Ground X, LG, 한화, 신한은행 등이 있으며, Sushiswap, 1inch, Klaytn, Badger와 같은 글로벌 블록체인 프로젝트와도 협업한 바 있습니다. 지금까지 약 300여곳 이상의 클라이언트를 대상으로 가장 신뢰할 수 있는 스마트 컨트랙트 보안감사 및 개발 서비스를 제공하였습니다.

문의 : audit@haechi.io

웹사이트 : audit.haechi.io

INTRODUCTION

본 보고서는 Krome 팀이 제작한 스마트 컨트랙트의 보안을 감사하기 위해 작성되었습니다. HAECHI AUDIT 은 Krome 팀이 제작한 스마트 컨트랙트의 구현 및 설계가 공개된 자료에 명시한 것처럼 잘 구현이 되어있고, 보안상 안전한지에 중점을 맞춰 감사를 진행했습니다.

CRITICAL

Critical 이슈는 광범위한 사용자가 피해를 볼 수 있는 치명적인 보안 결점으로 반드시 해결해야 하는 사항입니다.

MAJOR

Major 이슈는 보안상에 문제가 있거나 의도와 다른 구현으로 수정이 필요한 사항입니다.

MINOR

Minor 이슈는 잠재적으로 문제를 발생시킬 수 있으므로 수정이 요구되는 사항입니다.

TIPS

Tips 이슈는 수정했을 때 코드의 사용성이나 효율성이 더 좋아질 수 있는 사항입니다.


HAECHI AUDIT은 Krome 팀이 발견된 모든 이슈에 대하여 개선하는 것을 권장합니다. 이어지는 이슈 설명에서는 코드를 세부적으로 지칭하기 위해서 {파일 이름}#{줄 번호}, {컨트랙트 이름}#{함수/변수 이름} 포맷을 사용합니다. 예를 들면, *Sample.sol:20*은 Sample.sol 파일의 20번째 줄을 지칭하며, *Sample#fallback()* 는 Sample 컨트랙트의 fallback() 함수를 가리킵니다. 보고서 작성을 위해 진행된 모든 테스트 결과는 Appendix에서 확인 하실 수 있습니다.

SUMMARY

Audit에 사용된 코드는 GitHub (<https://github.com/krome-finance/krome-contract>)에서 찾아볼 수 있습니다. Audit에 사용된 코드의 마지막 커밋은 “53bc754c37876e667483c4930353283be1555577”입니다.

Issues HAECHI AUDIT에서는 Critical 이슈 0개, Major 이슈 1개, Minor 이슈 0개를 발견하였으며 수정했을 때 코드의 사용성이나 효율성이 더 좋아질 수 있는 사항들을 0개의 Tips 카테고리로 나누어 서술하였습니다.

Update [v.2.0] - 새로운 커밋
“b15e4b3e2f59b57737dddf6247a82290c314a753” 에서 Major 이슈 1개가 수정되었습니다,

| Severity | Issue | Status |
|--|---|-------------------------------------|
|  MAJOR | UsdkPriceOracle#getLatestPrice() 함수가 적절하지 않은 값을 반환할 수 있습니다. | (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

UsdkPriceOracle#getLatestPrice() 함수가 적절하지 않은 값을 반환할 수 있습니다. (Found - v.1.0) (Resolved - v.2.0)

```
// price0 = reserve1/reserve0 3:1 = 3
// in USD e9

function getLatestPrice() external view override returns (uint256) {
    // UQ112x112 → 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

UsdkPriceOracle#getLatestPrice() 함수에서 usdPerKlay의 decimal이 9가 아닌 경우가 존재할 수 있습니다. 이 경우, *UsdkPriceOracle#getLatestPrice()* 함수가 비정상적으로 크거나, 작은 값을 반환할 가능성이 존재합니다.

Recommendation

UsdkPriceOracle#getLatestPrice() 함수의 return 문을 KromePriceOracle#getLatestPrice() 함수와 비슷하게 decimal을 고려하는 형태로 수정하시는 것을 추천드립니다.

Update

[v2.0] - *UsdkPriceOracle#getLatestPrice()* 함수가 decimal을 고려하는 형태로 수정되어 문제가 해결되었습니다.

DISCLAIMER

해당 리포트는 투자에 대한 조언, 비즈니스 모델의 적합성, 버그 없이 안전한 코드를 보증하지 않습니다. 해당 리포트는 알려진 기술 문제들에 대한 논의의 목적으로만 사용됩니다. 리포트에 기술된 문제 외에도 메인넷상의 결함 등 발견되지 않은 문제들이 있을 수 있습니다. 안전한 스마트 컨트랙트를 작성하기 위해서는 발견된 문제들에 대한 수정과 충분한 테스트가 필요합니다.

Appendix A. Test Results

아래 결과는, 보안 감사 대상인 스마트 컨트랙트의 주요 로직을 커버하는 unit test 결과입니다.
붉은색으로 표시된 부분은 이슈가 존재하여 테스트에 통과하지 못한 테스트 케이스입니다.

```
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 KROME 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