SALUS SECURITY

JUNE 2023

CODE SECURITY ASSESSMENT

KINZA FINANCE

Overview

Project Summary

- Name: Kinza Finance
- Version: commit 68c100e
- Platform: BNB Smart Chain
- Language: Solidity
- Repository: https://github.com/Kinza-Finance/KZA-1.0
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Kinza Finance
Version	v2
Type	Solidity
Date	June 9 2023
Logs	May 31 2023; June 9 2023

Vulnerability Summary

Total High-Severity issues	3
Total Medium-Severity issues	0
Total Low-Severity issues	3
Total informational issues	14
Total	20

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Removed pool could lead to DOS	High	Business Logic	Resolved
2	Removed pool could lead to emission allocation errors and lock in the Minter contract	High	Business Logic	Resolved
3	Wrong logic in array pop	High	Business Logic	Resolved
4	removeVesting() is not removing the vester from vesters	Low	Business Logic	Resolved
5	Revote is missing in cancelRedeem()	Low	Code Consistency	Resolved
6	Incorrect configuration	Low	Configuration	Resolved
7	The pool can update rewards twice within a short period of time using the same voting weights.	Informational	Business Logic	Resolved
8	ERC20 return values not checked	Informational	Code Quality	Resolved
9	Missing validations	Informational	Code Quality	Partially Resolved
10	Can add a local variable instead of reading a storage multiple times	Informational	Gas Optimization	Resolved
11	Storages could be packed	Informational	Gas Optimization	Resolved
12	Can use immutable to save gas	Informational	Gas Optimization	Resolved
13	Duplicated codebase	Informational	Code Quality	Resolved
14	Code optimization	Informational	Code Quality	Partially Resolved
15	Duration hardcoded	Informational	Code Quality	Resolved
16	Redundant code	Informational	Code Quality	Partially Resolved
17	Improve readability	Informational	Code Quality	Resolved
18	Missing an error message	Informational	Code Quality	Resolved

19	Incorrect comments	Informational	Code Quality	Resolved
20	Typo	Informational	Code Quality	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Removed pool could lead to DOS

Severity: High	Category: Business Logic
Target: - src/contracts/KZA/Voter.sol	

Description

When a pool becomes inactive, the project team invokes the 'removeUnderlying' function to delete the pool information from 'bribes[]'. However, voting for a non-existent pool is not allowed in the voting logic. This will impact all balance-related operations in XKZA, as they all utilize the 'revote' function in the 'voter'.

If a pool is removed, it will result in a denial-of-service (DOS) situation for users who have voted on that pool concerning any XKZA token operations. The following call will revert due to the value of bribes[_pool] is zero.

- src/contracts/KZA/Voter.sol: L279

```
IBribe(bribes[_pool])._deposit(uint256(_poolWeight), _account);
```

- **Proof of Concept**

```
function testDOS() public {
    BaseSetup.setupVoter(100, 100);
    BaseSetup.doublePoolVote(alice, alice, 50, 50);
    address underlying = mp.popReserve();

    vm.prank(GOV);
    voter.removeUnderlying(underlying);
    address[] memory reserve = minter.getReserves();
    assertEq(reserve.length, 1);

    vm.expectRevert();
    vm.prank(alice);
    xkza.convert(10);

    vm.expectRevert();
    vm.prank(alice);
    xkza.redeem(10, 20 days);
}
```

Recommendation

Consider using try-catch to wrap the revote invocation and perform additional handling in case the revote call fails.

- **Example**

```
function convert(uint256 _amount) external nonReentrant returns(bool isRevote){
    _convert(_amount, msg.sender);

    try voter.reVote(msg.sender) {
        isRevote = true;
    } catch {
        isRevote = false;
    }
}
```

Status

This issue has been resolved by the team in PR #34.

2. Removed pool could lead to emission allocation errors and lock in the Minter contract

Severity: High	Category: Business Logic

Target:
- src/contracts/KZA/Minter.sol
- src/contracts/KZA/Voter.sol

Description

Even if a pool is removed from the Voter contract, the weight of the pool is included in totalWeight.

This will lead to the emission for this part being locked in the Minter contract and will affect the allocation mechanism of the emission.

- src/contracts/KZA/Minter.sol: L99 - L127

```
function update_period() external returns (uint lastEpoch) {
    lastEpoch = epoch;
    uint current = block.timestamp / WEEK;
    require(address(voter) != address(0), "voter needs to be set");
    require(current > lastEpoch, "only trigger each new week");
    epoch = current;
    KZA.mint(address(this), emission);
    uint256 prevEmission = emission;
    emission = (emission * (PRECISION - decay)) / PRECISION;
    // get the scheduled total
    address[] memory reserves = getReserves();
    uint256 length = reserves.length;
    if (length != 0) {
        address market;
        uint256 reward;
        uint256 totalWeight = voter.totalWeight();
        if (totalWeight != 0) {
            for (uint i; i < length;) {
            market = reserves[i];
            uint256 vote = voter.weights(market);
            reward = prevEmission * vote / totalWeight;
            rewardsCache[market] += reward;
            unchecked {
                ++i;
                }
            }
        }
    }
}
```

- **Proof of Concept**

```
function testLocked() public {
    BaseSetup.setupVoter(50, 50);
    BaseSetup.doublePoolVote(alice, alice, 50, 50);
```

```
    address underlying = mp.popReserve();
    vm.prank(GOV);
    voter.removeUnderlying(underlying);
    address[] memory reserve = minter.getReserves();
    assertEq(reserve.length, 1);

    skip(DURATION + 1);
    uint256 emission = minter.emission();
    minter.update_period();
    minter.notifyReward(USDC);

    //At this moment, there is only one pool for USDC, and all emissions should be
allocated to USDC.
    //The Minter contract should not include the KZA token.
    assertEq(kza.balanceOf(address(minter)), 0);
    assertEq(kza.balanceOf(address(rv)), emission);
}
```

Recommendation

Consider removing the weights associated with the pool when deleting it.

Status

This issue has been resolved by the team in PR #42.

3. Wrong logic in array pop

Severity: High	Category: Business Logic
Target: - src/contracts/KZA/Voter.sol	

Description

In the removeUnderlying() function, the last index of the markets array is removed. _underlying to be removed will still exist in the array.

- src/contracts/KZA/Voter.sol: L153 - L163

```
for (uint256 i; i < l;) {
        if (markets[i] == _underlying) {
            markets[l-1] = markets[i];
            markets.pop();
            emit MarketBribeRemoved(_underlying);
            return;
        }
        unchecked {
            ++i;
        }
}
```

Recommendation

Consider changing `markets[l-1] = markets[i];` to `markets[i] = markets[l-1];`

Status

This issue has been resolved by the team in PR #34.

4. removeVesting() is not removing the vester from vesters

Severity: Low	Category: Business Logic
Target: - src/contracts/KZA/VestingEscrow.sol	

Description

In the removeVesting() function, the vester should also be removed from the vesters array.

- src/contracts/KZA/VestingEscrow.sol: L117-L130

```
function removeVesting(address _vester) external onlyOwner {
   AccountInfo memory info = accountInfos[_vester];
   uint256 claimed = withdrawals[_vester];
   require(info.total != 0, "non existent vesting position");

   delete accountInfos[_vester];
   withdrawals[_vester] = 0;
   distributed -= info.total;
   distributed += claimed;
   claimedFromRemovedVesters += claimed;
   emit LogRemoveVest(_vester);
 }
```

Recommendation

Consider removing the vester from the array in the removeVesting() function.

Status

This issue has been resolved by the team in PR #38.

5. Revote is missing in cancelRedeem()

Severity: Low	Category: Code Consistency
Target: - src/contracts/KZA/XKZA.sol	

Description

In the XKZA contract, all functions that modify voting weights invoke the 'revote' function. However, the 'cancelRedeem' function does not have the 'revote' logic

- src/contracts/KZA/XKZA.sol: L292 - L303

```
function cancelRedeem(uint256 redeemIndex) external nonReentrant
validateRedeem(msg.sender, redeemIndex) {
    RedeemInfo storage _redeem = userRedeems[msg.sender][redeemIndex];

    // make redeeming xKZA available again
    userReedemTotal[msg.sender] -= _redeem.xAmount;
    _transfer(address(this), msg.sender, _redeem.xAmount);

    emit CancelRedeem(msg.sender, _redeem.xAmount);

    // remove redeem entry
    _deleteRedeemEntry(redeemIndex);
}
```

Recommendation

Consider calling revote at the end of the cancelRedeem function.

Status

This issue has been resolved by the team in PR #21.

6. Incorrect configuration

Severity: Low	Category: Configuration
Target: - src/contracts/KZA/Minter.sol	

Description

According to the inline comment, the value of decay should be 50 (0.5%).

- src/contracts/KZA/Minter.sol: L43

```
uint public decay = 100; // 0.5% weekly decay
```

Recommendation

Consider updating the value of decay.

Status

This issue has been resolved by the team in PR #22. The comment has been updated to align with the code.

2.3 Informational Findings

7. The pool can update rewards twice within a short period of time using the same voting weights.

Severity: Informational	Category: Business Logic
Target: - src/contracts/KZA/Minter.sol	

Description

The function only restricts two updates from occurring within the same epoch, which allows two updates to happen at the end of one epoch and the beginning of the next epoch.

This design flaw can be exploited by a liquidity pool, for example, Pool A, to bribe voters to vote for it at the end of an epoch. This enables Pool A to call update_period both at the end of the epoch and right at the beginning of the next epoch, causing votes with the same weight to be rewarded twice.

- src/contracts/KZA/Minter.sol: L99 - L127

```
function update_period() external returns (uint lastEpoch) {
    lastEpoch = epoch;
    uint current = block.timestamp / WEEK;
    require(address(voter) != address(0), "voter needs to be set");
    require(current > lastEpoch, "only trigger each new week");
    epoch = current;
    KZA.mint(address(this), emission);
    uint256 prevEmission = emission;
    emission = (emission * (PRECISION - decay)) / PRECISION;
    // get the scheduled total
    address[] memory reserves = getReserves();
    uint256 length = reserves.length;
    ...
}
```

Recommendation

Consider changing the condition current > lastEpoch to block.timestamp >= _period + WEEK, where _period represents the time of the previous update.

Status

This issue has been resolved by the team in PR #32. The team added a sync() function and an onlyEpochSynced() modifier to prevent users from doing vote-related activities until update_period() is called in an epoch.

8. ERC20 return values not checked

Severity: Informational	Category: Code Quality
Target: - src/contracts/KZA/KZADistributor.sol	

Description

The notifyReward function uses transferFrom of Reward token even if the contract sets SafeERC20 for IERC20.

- src/contracts/KZA/KZADistributor.sol: L136

```
REWARD.transferFrom(minter, vault, _amount);
```

Recommendation

Consider using OpenZeppelin's SafeERC20 versions with safeTransferFrom function that handle the return value check as well as non-standard-compliant tokens.

Status

This issue has been resolved by the team in PR #31.

9. Missing validations

Severity: Informational	Category: Code Quality

Target:
- src/contracts/KZA/XKZA.sol
- src/contracts/KZA/KZA.sol
- src/contracts/KZA/Voter.sol

Description

There are multiple places where zero address validation is missing.

- src/contracts/KZA/XKZA.sol: L180 - L183

```
function updateVoter(address _newVoter) external onlyOwner {
    voter = IVoter(_newVoter);
    emit NewVoter(_newVoter);
}
```

- src/contracts/KZA/XKZA.sol: L104 - L108

```
constructor(address _KZA, address _governance) {
    KZA = IKZA(_KZA);
    _transferWhitelist.add(address(this));
    transferOwnership(_governance);
}
```

- src/contracts/KZA/KZA.sol: L63 - L65

```
constructor(address _governance) {
    governance = _governance;
}
```

- src/contracts/KZA/KZA.sol: L69 - L73

```
function proposeNewGovernance(address _newGovernace) onlyGov external {
  newGovernanceProposedTime = block.timestamp;
  newGovernance = _newGovernace;
  emit NewGovernanceProposal(_newGovernace);
}
```

- src/contracts/KZA/KZA.sol: L85 - L88

```
function setBribeMinter(address _minter) onlyGov external {
    minter = _minter;
    emit NewBribeMinter(_minter);
}
```

- src/contracts/KZA/Voter.sol: L112 - L118

```
constructor(address _xToken, address _voteLogic, address  _bribeAssetRegistry, address
_governance) {
    xToken = _xToken;
    voteLogic = IVoteLogic(_voteLogic);
    bribeAssetRegistry = _bribeAssetRegistry;
    transferOwnership(_governance);
    emit NewVoteLogic(_voteLogic);
}
```

- src/contracts/KZA/Voter.sol: L211 - L214

```
function updateDelegate(address _delegatee) external {
    delegation[msg.sender] = _delegatee;
    emit SetDelegation(msg.sender, _delegatee);
}
```

Recommendation

Consider adding zero address validations.

Status

This issue has been partially resolved by the team in PR #37.

10. Can add a local variable instead of reading a storage multiple times

Severity: Informational	Category: Gas Optimization

Target:
- src/contracts/KZA/XKZA.sol
- src/contracts/KZA/KZA.sol
- src/contracts/KZA/Minter.sol
- src/contracts/KZA/VestingEscrow.sol
- src/contracts/KZA/KZADistributor.sol

Description

There are multiple places where memory can be used instead of storage.

- src/contracts/KZA/XKZA.sol: L272 - L277

  _redeem.xAmount is read twice from storage.

```
RedeemInfo storage _redeem = userRedeems[msg.sender][redeemIndex];
require(_currentBlockTimestamp() >= _redeem.endTime, "finalizeRedeem: vesting
duration has not ended yet");

// remove from SBT total
userReedemTotal[msg.sender] -= _redeem.xAmount;
_finalizeRedeem(msg.sender, _redeem.xAmount, _redeem.amount);
```

- src/contracts/KZA/XKZA.so:l L293 - L299

  _redeem.xAmount is read twice from storage.

```
RedeemInfo storage _redeem = userRedeems[msg.sender][redeemIndex];

// make redeeming xKZA available again
userReedemTotal[msg.sender] -= _redeem.xAmount;
_transfer(address(this), msg.sender, _redeem.xAmount);

emit CancelRedeem(msg.sender, _redeem.xAmount);
```

- src/contracts/KZA/KZA.sol: L76

  acceptNewGovernance() function reads newGovernance twice from storage.

```
function acceptNewGovernance() onlyNewGov external {
    require(block.timestamp > governanceDelay + newGovernanceProposedTime,
"pending governance delay");
    emit NewGovernance(governance, newGovernance);
    governance = newGovernance;
    newGovernance = address(0);
}
```

- src/contracts/KZA/Minter.sol: L105-L107

  For this, prevEmission could be set before minting.

```
KZA.mint(address(this), emission);
uint256 prevEmission = emission;
emission = (emission * (PRECISION - decay)) / PRECISION;
```

- src/contracts/KZA/VestingEscrow.sol: L228-L232

  For this, could use claimed instead of +=.

```
uint256 claimed = withdrawals[_vester];
if ((claimable - claimed) < _amount) {
    _amount = claimable - claimed;
}
withdrawals[_vester] += _amount;
```

- src/contracts/KZA/KZADistributor.sol: L125

  The notifyReward function reads the vault multiple times from storage.

```
function notifyReward(address _market, uint256 _amount) external onlyMinter {
    // if vault is not set, this would block the notifyRewardCall
    require(vault != address(0), "vault needs to be set");
    require(address(emisisonManager) != address(0), "emisisonManager needs to be
set");
    if (_amount != 0 ) {
        uint256 amountDToken = _amount * DTokenRatio() / PRECISION;
        uint256 amountAToken = _amount - amountDToken;

        uint256 DTokenVariable = amountDToken * variableDebtTokenRatio() /
PRECISION;
        uint256 DTokenStable = amountDToken * stableDebtTokenRatio / PRECISION;

        REWARD.transferFrom(minter, vault, _amount);
        // so transferStrategy can pull this amount in total through
increaseAllowance.
        IVault(vault).approveTransferStrat(_amount);
    ...

}
```

Recommendation

Consider using a local variable.

Status

This issue has been resolved by the team in PR #39 and PR #41.

11. Storages could be packed

Severity: Informational	Category: Gas Optimization
Target: - src/contracts/KZA/KZA.sol	

Description

Storage variables could be packed by moving initialMinted after newGovernanceProposedTime.

- src/contracts/KZA/KZA.sol: L27 - L32

```
bool public initialMinted;
uint public newGovernanceProposedTime;

address public minter;
address public governance;
address public newGovernance;
```

Recommendation

Consider moving initialMinted.

Status

This issue has been resolved by the team in PR #24.

12. Can use immutable to save gas

Severity: Informational	Category: Gas Optimization
Target: - src/contracts/KZA/KZADistributor.sol - src/contracts/KZA/Voter.sol	

Description

The following variables could be set immutable.

- src/contracts/KZA/KZA.sol: L46

```
IPool public pool;
```

- src/contracts/KZA/Voter.sol: L35

```
address public bribeAssetRegistry;
```

Recommendation

Consider changing to immutable.

Status

This issue has been resolved by the team in PR #30.

13. Duplicated codebase

Severity: Informational	Category: Code Quality

Target:
- src/contracts/KZA/KZADistributor.sol

Description

Each if case has a duplicated codebase.

- src/contracts/KZA/KZADistributor.sol: L146-L166

```
if (DTokenVariable != 0) {
    token = getReserveData(_market).variableDebtTokenAddress;
    rate = DTokenVariable / REWARD_PERIOD;
    emisisonManager.setDistributionEnd(token, address(REWARD),
uint32(block.timestamp + REWARD_PERIOD));
    rates[0] = rate.toUint88();
    emisisonManager.setEmissionPerSecond(token, rewards, rates);
}
if (DTokenStable != 0) {
    token = getReserveData(_market).stableDebtTokenAddress;
    rate = DTokenStable / REWARD_PERIOD;
    emisisonManager.setDistributionEnd(token, address(REWARD),
uint32(block.timestamp + REWARD_PERIOD));
    rates[0] = rate.toUint88();
    emisisonManager.setEmissionPerSecond(token, rewards, rates);
}
if (amountAToken != 0) {
    token = getReserveData(_market).aTokenAddress;
    rate = amountAToken / REWARD_PERIOD;
    emisisonManager.setDistributionEnd(token, address(REWARD),
uint32(block.timestamp + REWARD_PERIOD));
    rates[0] = rate.toUint88();
    emisisonManager.setEmissionPerSecond(token, rewards, rates);
}
```

Recommendation

Consider adding an internal function with token, rewards, and rate.

Status

This issue has been resolved by the team in PR #33.

14. Code optimization

Severity: Informational	Category: Code Quality

Target:
- src/contracts/KZA/AggregateBribe.sol
- src/contracts/KZA/VestingEscrow.sol
- src/contracts/KZA/Voter.sol
- src/contracts/KZA/KZADistributor.sol

Description

There are multiple places which codes can be optimized.

- src/contracts/KZA/AggregateBribe.sol: L213-L229

  _prevSupply could be inside if condition and not need to be set 1 because it's updated in if condition.

```
function earned(address token, address account) public view returns (uint) {
    ...
    uint _prevSupply = 1;

    if (_endIndex > 0) {
    for (uint i = _startIndex; i <= _endIndex - 1; i++) {
    Checkpoint memory cp0 = checkpoints[account][i];
    uint _nextEpochStart = _bribeStart(cp0.timestamp);
    // check that you've earned it
    // this won't happen until a week has passed
    if (_nextEpochStart > prevRewards.timestamp) {
    reward += prevRewards.balanceOf;
    }

    prevRewards.timestamp = _nextEpochStart;
    _prevSupply = supplyCheckpoints[getPriorSupplyIndex(_nextEpochStart +
DURATION)].supply;
    prevRewards.balanceOf = cp0.balanceOf *
tokenRewardsPerEpoch[token][_nextEpochStart] / _prevSupply;
    }
    ...
}
```

- src/contracts/KZA/VestingEscrow.sol: L117-L120

  Assigning claimed could be moved after checking info.total.

```
function removeVesting(address _vester) external onlyOwner {
    AccountInfo memory info = accountInfos[_vester];
    uint256 claimed = withdrawals[_vester];
    require(info.total != 0, "non existent vesting position");
    ...
}
```

- src/contracts/KZA/Voter.sol: L274-L278

  No need += because the initial value of votes[_account][_pool] is zero.

```
function _vote(address _account, address[] memory _poolVote, uint256[] memory
_weights) internal {
        ...
    for (uint i; i < _poolCnt;) {
    _pool = _poolVote[i];
    // _poolWeight is the actual weight, xToken 1 : 1
    _poolWeight = _weights[i] * _weight / _totalVoteWeight;
    // sanity check, it's always true given the _reset executes prior
    require(votes[_account][_pool] == 0, "non-zero existing vote");
    // a _weight of 0 should NOT be passed to this function
    require(_poolWeight != 0, "zero pool weight");
    poolVote[_account].push(_pool);

    weights[_pool] += _poolWeight;
    votes[_account][_pool] += _poolWeight;
    ...
}
```

- src/contracts/KZA/Voter.sol: L299-L303

  No need -= because the calculated result here will always be 0.

```
uint256 _votes = votes[_account][_pool];

if (_votes != 0) {
    weights[_pool] -= _votes;
    votes[_account][_pool] -= _votes;
```

- src/contracts/KZA/KZADistributor.sol: L133-L134

  DTokenStable could be calculated by reducing DTokenVariable from amountDToken.

```
uint256 DTokenVariable = amountDToken * variableDebtTokenRatio() / PRECISION;
uint256 DTokenStable = amountDToken * stableDebtTokenRatio / PRECISION;
```

Recommendation

Consider the recommendations provided.

Status

This issue has been partially resolved by the team in PR #40.

15. Duration hardcoded

Severity: Informational	Category: Code Quality
Target: - src/contracts/KZA/AggregateBribe.sol	

Description

7 days should be represented using the contract constant DURATION.

- src/contracts/KZA/AggregateBribe.sol: L118

```
return timestamp < bribeEnd ? bribeStart : bribeStart + 7 days;
```

- src/contracts/KZA/AggregateBribe.sol: L118

```
return timestamp - (timestamp % (7 days));
```

Recommendation

Consider changing 7 days to DURATION.

Status

This issue has been resolved by the team in PR #29.

16. Redundant code

Severity: Informational	Category: Code Quality

Target:
- src/contracts/KZA/KZADistributor.sol
- src/contracts/KZA/AggregateBribe.sol
- src/contracts/KZA/Voter.sol
- src/contracts/KZA/XKZA.sol

Description

- src/contracts/KZA/XKZA.sol: L30

  The SafeMath library is used to check underflow and overflow for arithmetic operations. However, since Solidity version 0.8.0, arithmetic operations revert on underflow and overflow by default.

  Because the XKZA contract uses a Solidity version no less than 0.8.0, there is no need to use the SafeMath library.

```
using SafeMath for uint256;
```

- src/contracts/KZA/KZADistributor.sol: L145

  No need to set token here.

```
token = getReserveData(_market).variableDebtTokenAddress;
if (DTokenVariable != 0) {
    token = getReserveData(_market).variableDebtTokenAddress;
    rate = DTokenVariable / REWARD_PERIOD;
    emisisonManager.setDistributionEnd(token, address(REWARD),
uint32(block.timestamp + REWARD_PERIOD));
    rates[0] = rate.toUint88();
    emisisonManager.setEmissionPerSecond(token, rewards, rates);
}
```

- src/contracts/KZA/AggregateBribe.sol: L115-L119

  The getEpochStart function is redundant as its calculation result is always the same as _bribeStart. It is recommended to delete it and use _bribeStart as a replacement.

```
function getEpochStart(uint timestamp) public pure returns (uint) {
    uint bribeStart = _bribeStart(timestamp);
    uint bribeEnd = bribeStart + DURATION;
    return timestamp < bribeEnd ? bribeStart : bribeStart + 7 days;
}
```

- src/contracts/KZA/Voter.sol: L103-L107

  onlyNewEpoch modifier is not used.

```
modifier onlyNewEpoch(address _xTokenHolder) {
    // ensure new epoch since last vote
    require((block.timestamp / DURATION) * DURATION > lastVoted[_xTokenHolder],
"holder already voted in this epoch");
    _;
}
```

Recommendation

Consider following the recommendations provided.

Status

This issue has been partially resolved by the team in PR #36.

17. Improve readability

Severity: Informational	Category: Code Quality
Target: - src/contracts/KZA/Minter.sol	

Description

7 days is more readable instead of 86400 * 7.

- src/contracts/KZA/Minter.sol: L32

```
uint internal constant WEEK = 86400 * 7;
```

Recommendation

Consider changing 86400 * 7 to 7 days.

Status

This issue has been resolved by the team in PR #28.

18. Missing an error message

Severity: Informational	Category: Code Quality
Target: - src/contracts/KZA/AggregateBribe.sol	

Description

The error message is missing even if _deposit function has.

- src/contracts/KZA/AggregateBribe.sol: L301

```
require(msg.sender == voter);
```

Recommendation

Consider adding the error message to fail more explicitly and ease debugging.

Status

This issue has been resolved by the team in PR #27.

19. Incorrect comments

Severity: Informational	Category: Code Quality

Target:
- src/contracts/KZA/AggregateBribe.sol
- src/contracts/KZA/XKZA.sol

Description

Incorrect comments.

- src/contracts/KZA/XKZA.sol: L188

  Comment should be max redeem duration.

```
/// @param _maxRedeemDuration min redeem duration
```

- src/contracts/KZA/AggregateBribe.sol: L266-L267

  Comment for ‘account’ should be the account to claim.

  Comment for ‘to’ should be the account to receive the reward.

```
/// @param account the reward token to claim
/// @param to the reward token to claim
```

Recommendation

Consider following the recommendations provided.

Status

This issue has been resolved by the team in PR #26.

20. Typo

Severity: Informational	Category: Code Quality
Target: - src/contracts/KZA/KZA.sol - src/contracts/KZA/Minter.sol	

Description

It's recommended to use the same name as ERC20 token name.

- src/contracts/KZA/KZA.sol: L14

```
contract KZA is ERC20("KINZA", "KZA"), ERC20Permit("Kinza") {
```

Typo for safeTransferFrom.

- src/contracts/KZA/Minter.sol: L163

```
// notifyReward would call safetTransferFrom
```

Recommendation

Consider using the same name as the ERC20 token name and fixing the typo in the comment.

Status

This issue has been resolved by the team in PR #25.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 68c100e:

File	SHA-1 hash
src/contracts/integration/RewardsVault.sol	9ff915a84c993adc7ecde1174d1b8138e7d5a486
src/contracts/integration/ReserveFeeDistributor.sol	9aba0ccd3d0c49a0ce76528370fe6606a41a8fca
src/contracts/integration/LockTransferStrategy.sol	973148fac0565bfe82e53ef6644795a4d1667717
src/contracts/integration/TransferStrategyBase.sol	d0bae95d9a9c6151302d8f1bf16a87fe5ccfd509
src/contracts/KZA/VoteLogic.sol	80b3962b3eb2aa58dd5f0197181868a48aa23b0f
src/contracts/KZA/KZADistributor.sol	f65140e261cc9f7a8b31b7ed39335c10cba5f310
src/contracts/KZA/VestingEscrow.sol	ba178d25d766034b9ebc11b3f8201b91d0c9752d
src/contracts/KZA/Voter.sol	e48bee8f9160af83dd75df0bb7d3c70cec9448dd
src/contracts/KZA/BribeAssetRegistry.sol	78a4ca2c3d1fbe0710bf9a1c8faaa5d29832b274
src/contracts/KZA/Minter.sol	ac376ba9636238a0bce00c780381bfc2cb488967
src/contracts/KZA/KZA.sol	34b0ed42bee4cc80da23da45d8414c702bdab25b
src/contracts/KZA/XKZA.sol	9e82d7c436a2c6c32c6dcd8c464c29bd132499e5
src/contracts/KZA/AggregateBribe.sol	3f3b726598e19e007296b9c496394b2785fe7446