SMART CONTRACT AUDIT REPORT

# Kinetix-fi

Version: 2.0

Protofire
November, 2023

## Table Of Contents

# The Kinetix-fi audit report

The audit report was prepared for the **Kinetix team** and includes following Github repository and files:

1) https://github.com/kinetixfi/kinetix-tokenomics/ audited commit fd32c3aaacf4c5d019aa5ad1569faca7ef2650ae:

   - /libraries/Math.sol
   - /libraries/Base64.sol
   - /factories/BribeFactory.sol
   - ExternalBribe.sol
   - VeArtProxy.sol
   - Voter.sol
   - VotingEscrow.sol

The code repository does not contain tests.

## Report Update - 1

The Kinetix team has updated its code in accordance with the issues identified in the first review.

The updated code was checked at the following commit:

https://github.com/kinetixfi/kinetix-tokenomics/

commit a73806ea8e20d2bf92a33f9ac460b9a9be80b033

## Procedure

The audit includes the following procedures:

- code analysis by the Slither static analyzer, followed by manual analysis and selection of problems
- manual code analysis, including logic check
- checking the business logic for compliance with the documentation (or white paper)

In addition, during the audit, we also provide recommendations on optimizing smart contracts and using best practices.

## Summary of audit findings

| Severity | Count |
|---|---|
| HIGH | 2 |
| MEDIUM | 2 |
| LOW | 3 |
| INFORMATIONAL | 9 |
| **TOTAL** | **16** |

## Severity classification

### High severity issues

High severity issues can lead to a direct or indirect loss of funds by both users and owners, a serious violation of the logic of the contract, including through the intervention of third parties. Such issues require immediate correction.

### Medium severity issues

Medium severity issues may cause contract functionality to fail or behave incorrectly. Also, medium severity issues can cause financial damage. Such issues require increased attention.

### Low severity issues

Low severity issues do not have a major security impact. It is recommended that such issues be taken into account.

### Informational severity issues

These issues provide general guidelines for writing code as well as best practices.

## Smart contracts VotingEscrow [KF-01]

The VotingEscrow contract implementation that escrows ERC-20 tokens in the form of an ERC-721 NFT.

The contract uses 1 year as max-lock time.

| ID: **GT01-01** | Severity: **High** | Status: **Fixed** |
|---|---|---|

### checkpoints.timestamp never updates

The `‘checkpoints’` mapping (L1100) has the Checkpoint structure with `‘timestamp’` and `‘tokenIds’` values. In the contract code the timestamp value never updates.

Thus, any function that uses the checkpoint’s `‘timestamp’` does not work as expected.

The functions with direct dependence on `‘timestamp’` are following:
`_findWhatCheckpointToWrite()`, `getPastVotesIndex()`

We highly recommend implement ‘checkpoints.timestamp’ updating functionality in the `_moveTokenDelegates()` and `_moveAllDelegates()` functions.

| ID: **KF01-02** | Severity: **Medium** | Status: **Fixed** |
|---|---|---|

## Approve doesn't work with merge() and withdraw()

It is assumed that users who have an approval can call the merge() or withdrow() function.

```
function _burn(uint _tokenId) internal {
      require(_isApprovedOrOwner(msg.sender, _tokenId), "caller is ...");
      address owner = ownerOf(_tokenId);
      // Clear approval
      approve(address(0), _tokenId);
      // checkpoint for gov
      _moveTokenDelegates(delegates(owner), address(0), _tokenId);
      // Remove token
      _removeTokenFrom(msg.sender, _tokenId);  // msg.sender!
      emit Transfer(owner, address(0), _tokenId);
}
```

But inside the merge() or withdrow() will be called _burn() and _removeTokenFrom() functions. And in such case the _removeTokenFrom() will use 'msg.sender' value as 'from' parameter. This will lead to failing transaction due to check on L491:

```
      assert(idToOwner[_tokenId] == _from);
```

Because the mapping 'idToOwner' only contains owner addresses.

Consider using 'owner' value instead of 'msg.sender' on L510.

|  |  |  |
|---|---|---|
| ID: **GT01-03** | Severity: **Low** | Status: **Fixed** |

## No revert messages

In almost all require statements there are no revert messages. Better to add simple messages for making debugging of the contract easier.

Also we recommend to use ‘`require`’ statement instead of ‘`assert`’ statements because in case of failing it will refund the remaining gas.

|  |  |  |
|---|---|---|
| ID: **GT01-04** | Severity: **Low** | Status: **Fixed** |

## Missing functionality

The functions `attach()` and `detach()` are never used in contract logic.

We recommend checking logic for these functions and (or) remove unused functions.

## Smart contract Voter [KF-02]

The contract is a part of a governance framework, leveraging voting mechanisms tied to token holdings to make decisions on governance-related actions within a protocol.

| | | |
|---|---|---|
| ID: **GT02-01** | Severity: **Medium** | Status: **Fixed** |

### **Everybody can set whitelisted addresses**

The function `initialize()` can be called by anyone and at any time. Because of that anybody can add any token to whitelist addresses.

This function should be disabled after the first call or used only by owner.

| | | |
|---|---|---|
| ID: **GT02-02** | Severity: **Low** | Status: **Fixed** |

### **Unreachable code**

The else statement of the `reset()` functions is never reached because the '`_votes`' value more than zero always.

*Check comments in the following code:*

```
function _reset(uint _tokenId) internal {
      ...
      for (uint i = 0; i < _poolVoteCnt; i ++) {
        address _pool = _poolVote[i];
        uint256 _votes = votes[_tokenId][_pool];

      if (_votes != 0) {      // uint value and != 0
        weights[_pool] -= _votes;
        votes[_tokenId][_pool] -= _votes;
        if (_votes > 0) {              // always true
          IBribe(external_bribes[_pool])._withdraw(uint256(_votes), _tokenId);
          _totalWeight += _votes;
        } else {
          _totalWeight -= _votes;    // unreachable
        }
        emit Abstained(_tokenId, _votes);
       }
      }
      totalWeight -= uint256(_totalWeight);
      ...
      }
```

We recommend checking logic and make sure this functions works as intended.

| ID: **GT02-03** | Severity: **Info** | Status: **Fixed** |
|---|---|---|

## Not indexed arguments

Better to have indexed arguments of events:

1. In the event `VoteablePoolAdded()` the argument creator isn't indexed
2. In the events Voted and Abstained the argument tokenId isn't indexed

| | | |
|---|---|---|
| ID: **GT02-04** | Severity: **Info** | Status: **Fixed** |

## No events

These functions doesn’t have appropriate events:

1. `setGovernor()`
2. `setEmergencyCouncil()`
3. `poke()`

Better to add events for all non-view functions.

|  |  |  |
|---|---|---|
| ID: **GT02-05** | Severity: **Info** | Status: **Fixed** |

## No revert messages

In almost all require statements there are no revert messages. Better to add simple messages for making debugging of the contract easier.

## Smart contract ExternalBribe [KF-03]

The contract is used to pay out rewards for a given pool based on the votes that were received from the users.

| ID: **KF03-01** | Severity: **High** | Status: **Fixed** |
|---|---|---|

### Reward tokens can be lost

There are two possible ways how reward tokens can be lost in the contract:

1. Tokens were sent to the contract using the `notifyRewardAmount()` function in the epoch in which at the end of it there was zero total supply. In this case nobody will be able to claim the rewards and they will be lost. There is no way to retrieve these tokens from the contract.
2. If a user (Bob) deposited in previous epochs but in the current epoch he didn't make any checkpoints using the `reset()` or the `poke()` or the `vote()` function in the Voter contract. All rewards from the current epoch will be splitted between all users of the contract and the total supply of the contract will include the balance of Bob, but he won't be able to claim his tokens because the function `earned()` don't include epochs in which a user didn't make a checkpoint. Also, other users from the current epoch won't be able to claim his rewards too and they will be lost on the contract.

The first case can be solved by adding a new function that will transfer rewards from an epoch in which there was a zero total supply at the end.

The second case can be solved by fixing the `earned()` function. It should include all epochs for a user, not only epochs in which the user made checkpoints.

| ID: **KF03-02** | Severity: **Info** | Status: **Not Fixed** |
|---|---|---|

## Possibility of denial of service with gas limit

The function `earned()` possibly may encounter denial of service with a gas limit for a user. It could happen in case the user didn’t call the `getReward()` function in the contract or the `claimBribes()` function in the Voter contract for a very long time.

| ID: **GT03-03** | Severity: **Info** | Status: **Not Fixed** |
|---|---|---|

## No revert messages

In almost all require statements there are no revert messages. Better to add simple messages for making debugging of the contract easier.

## Update

Not all issues have been fixed in the updated code.

|  |  |  |
|---|---|---|
| ID: **KF03-04** | Severity: **Info** | Status: **Fixed** |

## Magic numbers

On lines 77 and 83 there are magic numbers 7 days that should be replaced with the variable DURATION to increase readability.

| ID: **KF03-05** | Severity: **Info** | Status: **Fixed** |
|---|---|---|

## No events, no array-view functions

1. These functions doesn’t have appropriate events:
   - `earned()`,
   - `swapOutRewardToken()`,
   - adding rewards in the contract constructor

   Better to add events for all non-view functions.

2. Consider adding view function for the ‘rewards’ variable to fetch actual rewards list.

| ID: **KF03-06** | Severity: **Info** | Status: **Fixed** |
|---|---|---|

## Gas savings

1. Consider using `'calldata'` data structure instead of `'memory'` for the array function parameters.
2. The state variable `'PRECISION'` is never used and can be removed.
3. The `lastTimeRewardApplicable()` can be declared as `'external'`, since it never called inside the contract.

Since the project is going to be deployed on cheap network the severity of this issue marked as Info.

## Smart contract VeArtProxy [KF-04]

The contract is used to generate URLs.

**No issues have been identified.**

## Smart contract BribeFactory [KF-05]

The factory contract to deploy new ExternalBribe contracts.

**No issues have been identified.**

## Libraries Math, Base64 [KF-06]

The libraries for generating URL addresses.

| | | |
|---|---|---|
| ID: **KF06-01** | Severity: **Info** | Status: **Not Fixed** |

### Gas saving

The `max()`, `sqrt()`, `cbrt()` functions are never used in contracts code and can be removed to save gas on deployment.

### Update

Since the project is going to be deployed on 'cheap' network the significance of this issue is extremely small.

## Conclusion

During the audit, 2 high and 2 medium severity issues were found. However, the provided repository does not contain tests for contracts. Adding tests for project contracts is strongly recommended.

Furthermore, fixing issues and covering them with tests is strongly advised.

For the convenience of users and developers, we recommend adding NatSpec documentation documentation for all contract functions.

Please note that auditing is not a complete replacement for unit tests and integration tests.

## Conclusion update

All of the high, medium and low severity issues have been resolved in the code update. There are some minor issues remaining which we don’t think represent possible danger of tampering with the contract suite.

## Disclaimer

Note that smart contract audit provided by Protofire is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of 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. As one audit-based assessment cannot be considered comprehensive, Protofire recommends proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Smart contract audit provided by Protofire shall not be used as investment or financial advice.