Enjinstarter

Staking Technical Specifications

August 26, 2022

Overview

This project allows the company to create multiple staking pools using smart contracts for our users to earn rewards by locking up specific ERC20 tokens for a specified duration.

Functional Requirements

Roles

This project has the following 4 roles:

- **Default Admin**: Admin role that controls the granting of roles to and revoking roles from accounts
- **Governance**: Controls access to functions that may result in loss of funds such as the change of staking pool contract address, the change of admin wallet to receive revoked stakes and unused rewards and pause/unpause contract
- **Contract Admin**: Controls access to functions for day-to-day operations such as create staking pool, add staking pool reward, open/close staking pool, suspend/resume staking pool, suspend/resume stake and revoke stake
- **User**: Can stake funds, claim rewards and unstake funds in available staking pools

Features

This project has the following features:

- Change staking pool contract address (Governance)
- Change admin wallet to receive revoked stakes and unused rewards (Governance)
- Pause/unpause contract (Governance)
- Create staking pools (Contract Admin)
- Add staking pool rewards (Contract Admin)
- Open/close staking pool (Contract Admin)
- Suspend/resume staking pool (Contract Admin)
- Suspend/resume stake (Contract Admin)
- Revoke stake (Contract Admin)

- Withdraw revoked stakes to configured admin wallet (Contract Admin)
- Withdraw unused rewards to configured admin wallet (Contract Admin)
- Stake funds (User)
- Claim rewards earned (User)
- Unstake funds (User)

Use Cases

1. Contract admins create different staking pools containing the following information:
    a. Pool ID
    b. Stake Duration in Days
    c. Stake Token Address
    d. Stake Token Decimals
    e. Reward Token Address
    f. Reward Token Decimals
    g. Pool APR in Wei
2. Contract admins add rewards to the staking pool
3. Users are allowed to stake in staking pool as long as pool has sufficient rewards and pool is open
4. Users can only claim rewards and unstake funds after stake has matured as long as both pool and stake have not been suspended and stake has not been revoked
5. Users can increase stake for existing stake that has not matured yet. Stake maturity date will be reset while rewards earned up to the point where stake is increased will be accumulated
6. Contract admins can close staking pool to stop accepting new stakes for pool
7. Contract admins can suspend staking pool to prevent users from claiming rewards or unstaking funds from pool in case need to investigate irregularities
8. Contract admins can suspend stake of specific user to prevent specific user from claiming rewards or unstaking funds in case need to investigate irregularities
9. Contract admins can revoke stake of specific user as compensation
10. Contract admins can reopen closed staking pool to start accepting new stakes for pool
11. Contract admins can resume suspended staking pool to allow users to claim rewards or unstake funds

12. Contract admins can resume suspended stake of specific user to allow user to claim rewards or unstake funds
13. Contract admins can withdraw revoked stakes to configured admin wallet
14. Contract admins can withdraw unused rewards to configured admin wallet
15. Governance can change the admin wallet where revoked stakes and unused rewards are transferred to in case wallet has been compromised
16. Governance can change the staking pool contract address in case of error
17. Governance can pause and unpause contract in case of emergencies

Highly Permissive Functions

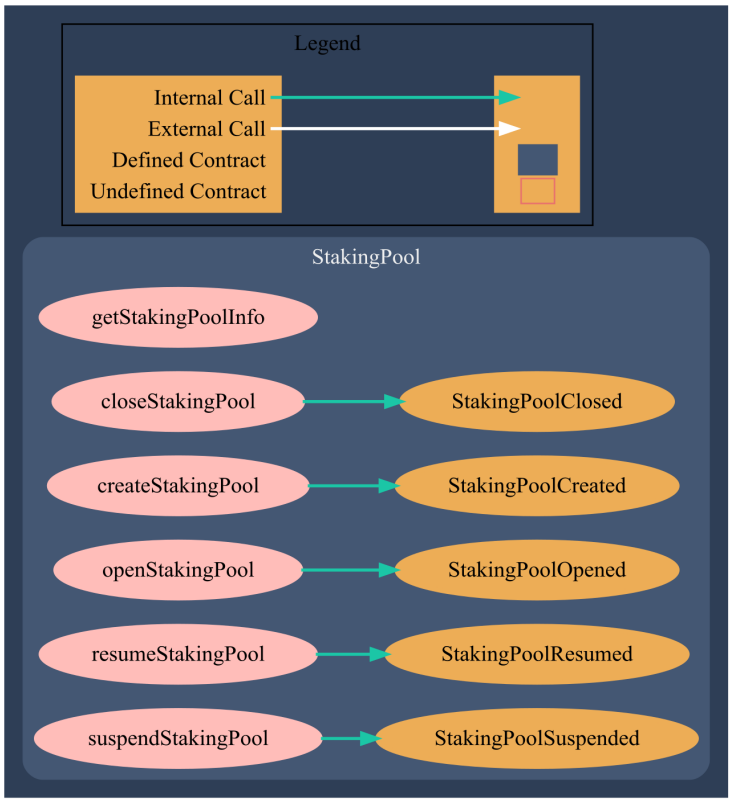
1. Governance can change the staking pool contract address
2. Governance can change the admin wallet
3. Governance can pause and unpause the contract
4. Contract Admin can close and reopen staking pools
5. Contract Admin can suspend and resume staking pools
6. Contract Admin can suspend and resume user stakes
7. Contract Admin can revoke user stakes
8. Contract Admin can withdraw revoked stakes to the admin wallet
9. Contract Admin can withdraw unused rewards to the admin wallet

There is a risk that an attacker can get access to funds that belong to users if governance fund keys are leaked.

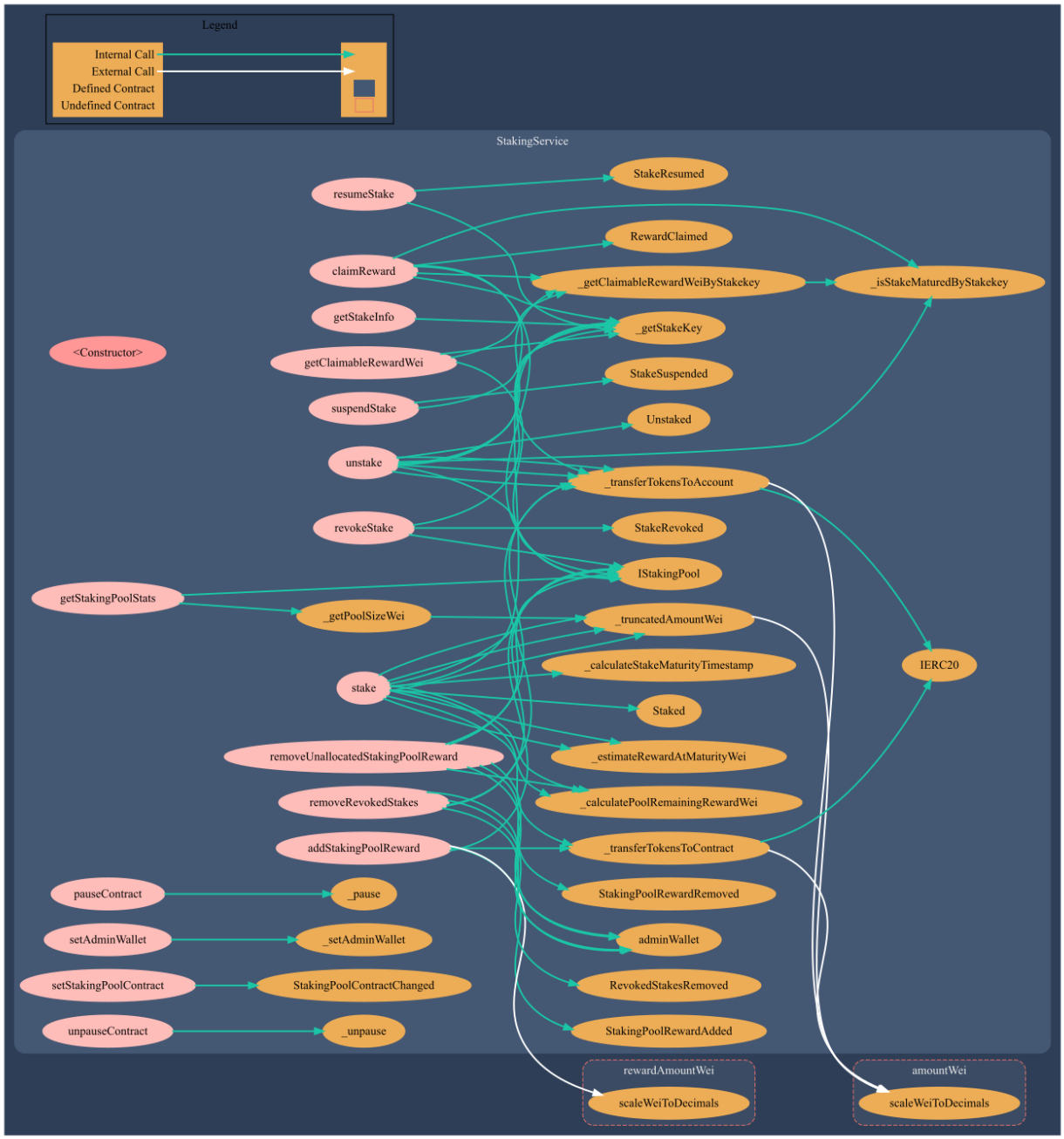
Technical Requirements

Architecture Overview

StakingPool



StakingService



Contract Information

This section contains detailed information (purpose, assets, functions and events) about the contracts used in this project.

AdminPrivileges.sol

Access control contract inherits from OpenZeppelin's AccessControl.sol and provides the role definitions that are inherited by other contracts.

Assets

The admin privileges contract contains the following 2 constants:

- GOVERNANCE_ROLE
- CONTRACT_ADMIN_ROLE

Functions

- constructor(): grants contract creator the 3 roles of Default Admin, Governance and Contract Admin

AdminWallet.sol

Admin wallet contract provides an implementation of the admin wallet interface that is inherited by other contracts.

Assets

The admin wallet contract contains the admin wallet entity that stores the address of the wallet that will receive the revoked stakes and unused rewards.

Events

The admin wallet contract has the following event:

- AdminWalletChanged: emitted when the admin wallet has been changed from oldWallet to newWallet

Functions

The admin wallet contract has the following 3 functions:

- constructor(): sets the contract creator as the admin wallet
- adminWallet(): returns the admin wallet address
- _setAdminWallet(address newWallet): internal function that sets the input parameter newWallet as the admin wallet

StakingPool.sol

Staking pool contract inherits from AdminPrivileges.sol and stores the staking pool configuration information that is used by the staking service contract.

Assets

Ths staking pool contract contains the following Struct:

- StakingPoolInfo: This structure contains the information about a staking pool
    - uint256 stakeDurationDays: stake duration in days
    - address stakeTokenAddress: stake token address
    - uint256 stakeTokenDecimals: stake token decimals
    - address rewardTokenAddress: reward token address
    - uint256 rewardTokenDecimals: reward token decimals
    - uint256 poolAprWei: staking pool APR in wei
    - bool isOpen: true if staking pool allows users to stake funds
    - bool isActive: true if staking pool allows users to claim rewards and unstake funds
    - bool isInitialized: true if staking pool has been initialized

The following 2 entities are present in the staking pool contract:

- _stakingPools: a private mapping of pool ID to StakingPoolInfo struct
- TOKEN_MAX_DECIMALS: maximum acceptable token decimals

Events

The staking pool contract has the following 5 events:

- StakingPoolClosed: emitted when staking pool has been closed by contract admin
- StakingPoolCreated: emitted when staking pool has been created by contract admin
- StakingPoolOpened: emitted when closed staking pool has been reopened by contract admin
- StakingPoolResumed: emitted when suspended staking pool has been resumed by contract admin
- StakingPoolSuspended: emitted when staking pool has been suspended by contract admin

Functions

The staking pool contract has the following 6 functions:

- getStakingPoolInfo(bytes32 poolId): returns the staking pool information for the specified pool ID.
- closeStakingPool(bytes32 poolId): allows contract admins to close the staking pool corresponding to the specified pool ID
- createStakingPool(bytes32 poolId, uint256 stakeDurationDays, address stakeTokenAddress, uint256 stakeTokenDecimals, address rewardTokenAddress, decimals rewardTokenDecimals, uint256 poolAprWei): allows contract admins to create staking pool with the specified pool information
- openStakingPool(bytes32 poolId): allows contract admins to reopen the closed staking pool corresponding to the specified pool ID
- resumeStakingPool(bytes32 poolId): allows contract admins to resume the suspended staking pool corresponding to the specified pool ID
- suspendStakingPool(bytes32 poolId): allows contract admins to suspend staking pool corresponding to the specified pool ID

StakingService.sol

Staking service contract stores the stake data and runtime staking pool data and inherits from OpenZeppelin's Pausable.sol, AdminPrivileges.sol and AdminWallet.sol.

Assets

The staking service contract contains the following 2 Structs:

- StakeInfo: This structure contains information about a stake
    - uint256 stakeAmountWei: total stake amount in wei
    - uint256 lastStakeAmountWei: last stake amount in wei
    - uint256 stakeTimestamp: last stake timestamp
    - uint256 stakeMaturityTimestamp: stake maturity timestamp
    - uint256 estimatedRewardAtMaturityWei: reward amount at stake maturity in wei
    - uint256 rewardClaimedWei: reward amount user has already claimed in wei
    - bool isActive: true if user is allowed to claim rewards and unstake funds
    - bool isInitialized: true if stake has been initialized
- StakingPoolStats: This structure contains the runtime staking pool data
    - totalRewardWei: total staking pool reward held by contract in wei
    - totalStakedWei: total funds staked inside staking pool in wei
    - rewardToBeDistributedWei: allocated staking pool reward to be distributed to users in wei
    - totalRevokedStakeWei: total revoked stake amount in wei

The following 7 entities are present in the staking service contract:

- stakingPoolContract: address of staking pool contract to retrieve staking pool configuration information
- _stakes: private mapping of abi.encode(address account, bytes32 poolId) to StakeInfo struct
- _stakingPoolStats: private mapping of pool ID to StakingPoolStats struct
- DAYS_IN_YEAR: 365 days in a year
- PERCENT_100_WEI: 100% in wei
- SECONDS_IN_DAY: 86400 seconds in a day
- TOKEN_MAX_DECIMALS: maximum acceptable token decimals

Events

The staking service contract has the following 10 events:

- RevokedStakesRemoved: emitted when revoked stakes have been removed from pool by contract admin
- RewardClaimed: emitted when reward has been claimed by user

- Staked: emitted when stake has been placed by user
- StakeResumed: emitted when suspended stake has been resumed by contract admin
- StakeRevoked: emitted when stake with reward has been revoked by contract admin
- StakeSuspended: emitted when stake has been suspended by contract admin
- StakingPoolContractChanged: emitted when staking pool contract address has been changed by governance
- StakingPoolRewardAdded: emitted when reward has been added to pool by contract admin
- StakingPoolRewardRemoved: emitted when unused reward has been removed from pool by contract admin
- Unstaked: emitted when matured stake with unclaimed reward has been withdrawn by user

Functions

The staking service contract has the following 27 functions:

- constructor(address stakingPoolContract_): sets the staking pool contract address to the input parameter value
- getClaimableRewardWei(bytes32 poolId, address account): returns the claimable reward in wei for the stake corresponding to the specified pool ID and account address
- getStakeInfo(bytes32 poolId, address account): returns the stake information for the stake corresponding to the specified pool ID and account address
- getStakingPoolStats(bytes32 poolId): returns the staking pool runtime data for the specified pool ID
- claimReward(bytes32 poolId): allows users to claim reward of their matured stake from the staking pool corresponding to the specified pool ID
- stake(bytes32 poolId, uint256 stakeAmountWei): allows users to stake specified amount in wei inside staking pool corresponding to the specified pool ID
- unstake(bytes32 poolId): allows users to unstake and claim their unclaimed reward of their matured stake from the staking pool corresponding to the specified pool ID

- addStakingPoolReward(bytes32 poolId, uint256 rewardAmountWei): allows contract admin to add specified reward amount in wei to the staking pool corresponding to the specified pool ID
- removeRevokedStakes(bytes32 poolId): allows contract admin to withdraw revoked stakes from staking pool corresponding to the specified pool ID to configured admin wallet
- removeUnallocatedStakingPoolReward(bytes32 poolId): allows contract admin to withdraw unused rewards from staking pool corresponding to the specified pool ID to configured admin wallet
- resumeStake(bytes32 poolId, address account): allows contract admin to resume the stake corresponding to the specified pool ID and account address
- revokeStake(bytes32 poolId, address account): allows contract admin to revoke the stake corresponding to the specified pool ID and account address
- suspendStake(bytes32 poolId, address account): allows contract admin to suspend the stake corresponding to the specified pool ID and account address
- pauseContract(): allows governance to pause contract
- setAdminWallet(address newWallet): allows governance to change admin wallet address
- setStakingPoolContract(address newStakingPool): allows governance to change staking pool contract address
- unpauseContract(): allows governance to unpause contract
- _getStakeKey(bytes32 poolId, address account): returns the stake key corresponding to the specified pool ID and account address
- _truncatedAmountWei(uint256 amountWei, uint256 tokenDecimals): returns the wei amount truncated to the specified token decimals
- _calculatePoolRemainingRewardWei(bytes32 poolId): returns the remaining reward in wei for staking pool corresponding to the specified pool ID
- calculateStakeMaturityTimestamp(uint256 stakeDurationDays, uint256 stakeTimestamp): returns the stake maturity timestamp
- _estimateRewardAtMaturityWei(uint256 stakeDurationDays, uint256 poolAprWei, uint256 stakeAmountWei): returns the reward at maturity in wei for the specified stake duration in days and pool APR in wei

- _getClaimableRewardWeiByStakekey(bytes memory stakekey): returns the claimable reward in wei for the stake corresponding to the specified stake key
- _getPoolSizeWei(uint256 stakeDurationDays, uint256 poolAprWei, uint256 stakeTokenDecimals): returns the staking pool size in wei truncated to stake token decimals for the specified stake duration in days and pool APR in wei
- _isStakeMaturedByStakekey(bytes memory stakekey): returns true if stake corresponding to the specified stake key has matured
- _transferTokensToAccount(address tokenAddress, uint256 tokenDecimals, uint256 amountWei, address account): transfer tokens from staking service contract to specified account address taking into account the token decimals
- _transferTokensToContract(address tokenAddress, uint256 tokenDecimals, uint256 amountWei, address account): transfer tokens from specified account address to staking service contract taking into account the token decimals

libraries/UnitConverter.sol

Unit converter library contract provides utility functions to convert between amounts specified in wei and decimals.

Functions

The unit converter library contract has the following 2 functions:

- scaleWeiToDecimals(uint256 weiAmount, uint256 decimals): returns the wei amount scaled down to decimals amount
- scaleDecimalsToWei(uint256 decimalsAmount, uint256 decimals): returns the decimals amount scaled up to wei amount