BEOSIN
Blockchain Security

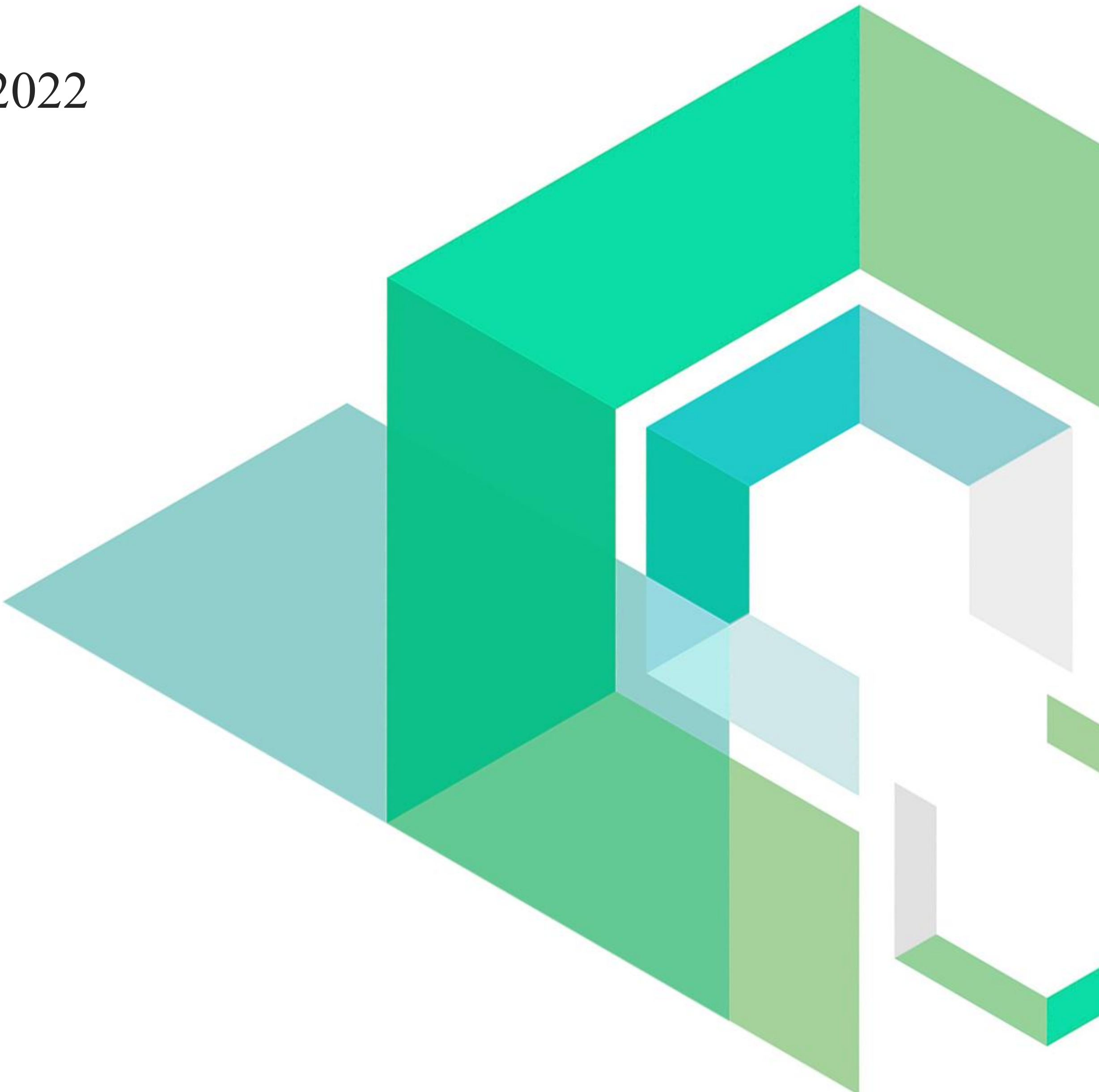
Ankr-staking

Smart Contract Security Audit

V1.0

No. 202212061649

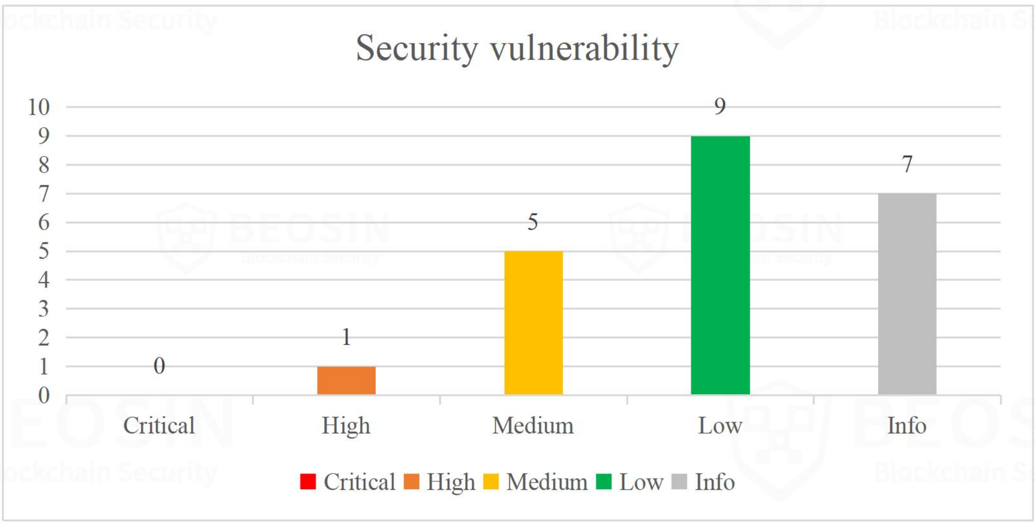
Dec 6th, 2022



Contents

Summary of Audit Results

After auditing, 1 High-risk item, 5 Medium-risk items, 9 Low-risk items and 7 Info items were identified in the Ankr-staking project. Specific audit details will be presented in the **Findings section**. Users should pay attention to the following aspects when interacting with this project：



***Notes:**

- **Risk Description:**

1. The AnkrProtocol contract does not implement the extraction function, the contract does not support users to withdraw staked assets.

2. Governance has the authority to modify the stake amount corresponding to the level to any value, resulting in user increase stake may decrease level.

3. The userDeposit.expires is controlled by the user and the assets is locked once.

4. The PayAsYouGo contract implements the final withdrawal operation by Consensus, and the handling fee is arbitrarily controlled by Consensus.

- **Project Description:**

1. Business overview

Ankr is a stake-type project. Users can spend the BEP20 token specified in the contract to register the address as a validator, and can set the validator's reward commission between 0% and 30%. When the validator is successfully registered, other users can spend the specified BEP20 to stake the validator. The user's reward amount in a specific epoch is the validator's reward amount multiplied by the user's stake ratio after deducting the owner's commission fee. It should be noted that the rewards need to be actively transferred to the validator. If there is no reward transfer, even if there is a staked amount, the rewards cannot be obtained.

1 Overview

1.1 Project Overview

Project Name	Ankr-staking
Platform	BNB Chain
Audit scope	https://github.com/Ankr-network/ankr-contracts/tree/STAKAN-00-undelegation_process

Commit Hash	5bc1483497c2846edcad6978396240e603a24a34(Initial)
	23469d9d83bcd39e2454324ec72a7d01d509f2fc(Latest)

1.2 Audit Overview

Audit work duration：Sep 14, 2022 – Dec 6, 2022

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
Ankr-staking-1	No funds transferred when registering validator	High	Fixed
Ankr-staking-2	AdvanceStakingRewards execution order issue	Medium	Fixed
Ankr-staking-3	Deleted validator user cannot extract	Medium	Fixed
Ankr-staking-4	Asset types are not uniform	Medium	Fixed
Ankr-staking-5	Delete staking records without rewards	Low	Fixed
Ankr-staking-6	The dustRewards reward cannot be withdrawn	Low	Fixed
Ankr-staking-7	Inaccurate principal withdrawal amount	Low	Fixed
Ankr-staking-8	Unreasonable *stake* function	Low	Fixed
Ankr-staking-9	Not updating change.at to global variable	Info	Fixed
Ankr-staking-10	Inconsistent accuracy	Info	Fixed
Ankr-staking-11	Variable types are inconsistent	Info	Fixed
Ankr-staking-12	Meaningless payable keyword	Info	Fixed
AnkrProtocol-1	Missing asset extraction interface	Medium	Acknowledged
AnkrProtocol-2	Stake level setting issue	Low	Acknowledged
AnkrProtocol-3	Missing check for 0 address	Low	Fixed
AnkrProtocol-4	Code redundancy	Low	Fixed
AnkrProtocol-5	Lack of event triggering	Info	Fixed
AnkrProtocol-6	The stake lock issue	Info	Acknowledged
PayAsYouGo-1	Reward source not specified	Medium	Fixed
PayAsYouGo-2	User withdrawal issue	Low	Acknowledged
PayAsYouGo-3	Handling fee issue	Low	Fixed
PayAsYouGo-4	Variable does not implement the relevant function	Info	Acknowledged

Status Notes:

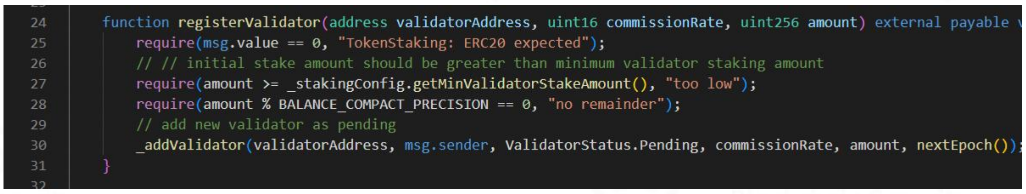
- AnkrProtocol-1 is unfixed and will cause no function to withdraw after the user stakes.
- AnkrProtocol-2 is unfixed and will cause after the governance modifies the stake amount of the corresponding level, the user's stake may not match the level.

- AnkrProtocol-6 is unfixed and will cause in the *_lockDeposit* function, the lockup time is only updated when it is judged that userDeposit.expires == 0. The user can control the lockup time and only lock the first stake.
- PayAsYouGo-2 is unfixed and will cause final lending operation is implemented by Consensus calling the *handleWithdraw* function. The user's withdrawal amount and Fee are not controlled by the user.
- PayAsYouGo-4 is unfixed and will not cause any issues.

Finding Details:

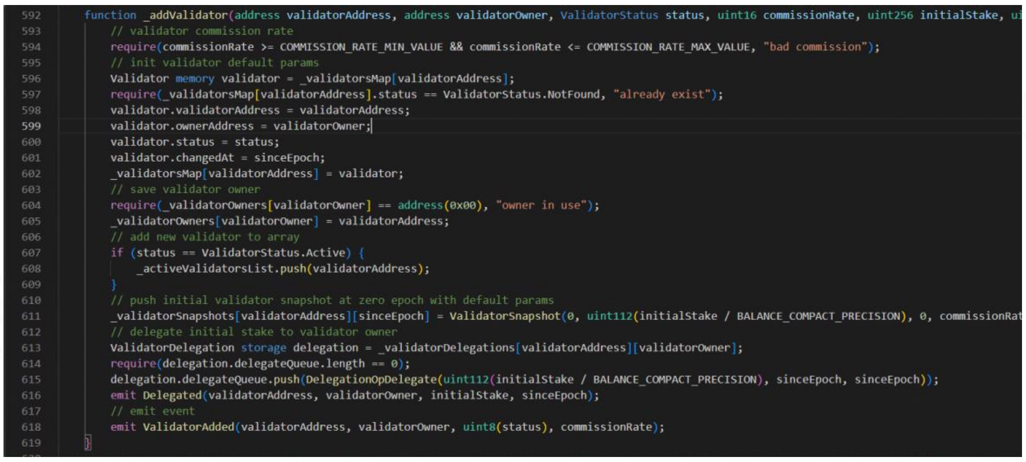
[Ankr-staking-1] No funds transferred when registering validator

Severity Level	High
Type	Business Security
Lines	TokenStaking.sol #L24-31
Description	When the validator is registered through the *registervalidator* function, it is not verified whether there is a token transfer corresponding to the amount. Then user can withdraw the assets in the contract by registering any number of stake records.



```
24      function registerValidator(address validatorAddress, uint16 commissionRate, uint256 amount) external payable
25          require(msg.value == 0, "TokenStaking: ERC20 expected");
26          // // initial stake amount should be greater than minimum validator staking amount
27          require(amount >= _stakingConfig.getMinValidatorStakeAmount(), "too low");
28          require(amount % BALANCE_COMPACT_PRECISION == 0, "no remainder");
29          // add new validator as pending
30          _addValidator(validatorAddress, msg.sender, ValidatorStatus.Pending, commissionRate, amount, nextEpoch())
31      }
```

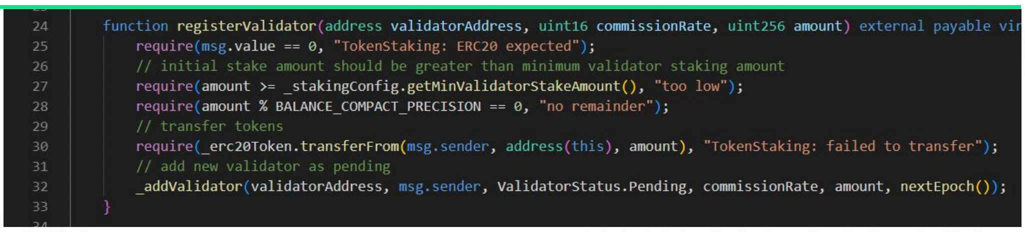
Figure 1 The source code of *registervalidator* function



```
function _addValidator(address validatorAddress, address validatorOwner, ValidatorStatus status, uint16 commissionRate, uint256 initialStake, ui
    // validator commission rate
    require(commissionRate >= COMMISSION_RATE_MIN_VALUE && commissionRate <= COMMISSION_RATE_MAX_VALUE, "bad commission");
    // init validator default params
    Validator memory validator = _validatorsMap[validatorAddress];
    require(_validatorsMap[validatorAddress].status == ValidatorStatus.NotFound, "already exist");
    validator.validatorAddress = validatorAddress;
    validator.ownerAddress = validatorOwner;
    validator.status = status;
    validator.changedAt = sinceEpoch;
    _validatorsMap[validatorAddress] = validator;
    // save validator owner
    require(_validatorOwners[validatorOwner] == address(0x00), "owner in use");
    _validatorOwners[validatorOwner] = validatorAddress;
    // add new validator to array
    if (status == ValidatorStatus.Active) {
        _activeValidatorsList.push(validatorAddress);
    }
    // push initial validator snapshot at zero epoch with default params
    _validatorSnapshots[validatorAddress][sinceEpoch] = ValidatorSnapshot(0, uint112(initialStake / BALANCE_COMPACT_PRECISION), 0, commissionRat
    // delegate initial stake to validator owner
    ValidatorDelegation storage delegation = _validatorDelegations[validatorAddress][validatorOwner];
    require(delegation.delegateQueue.length == 0);
    delegation.delegateQueue.push(DelegationOpDelegate(uint112(initialStake / BALANCE_COMPACT_PRECISION), sinceEpoch, sinceEpoch));
    emit Delegated(validatorAddress, validatorOwner, initialStake, sinceEpoch);
    // emit event
    emit ValidatorAdded(validatorAddress, validatorOwner, uint8(status), commissionRate);
}
```

Figure 2 The source code of *_addvalidator* function

Recommendations	It is recommended to verify whether there is a corresponding amount of tokens transferred in.
Status	Fixed.



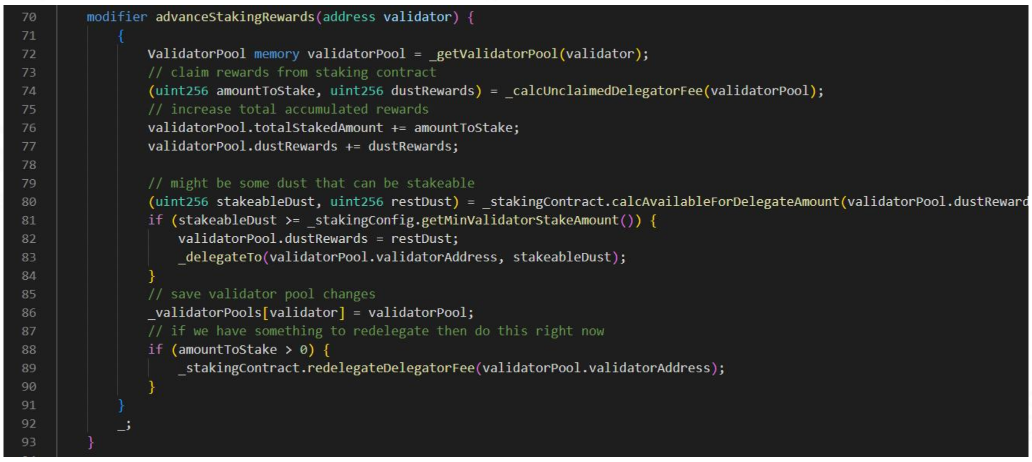
```
function registerValidator(address validatorAddress, uint16 commissionRate, uint256 amount) external payable vir
    require(msg.value == 0, "TokenStaking: ERC20 expected");
    // initial stake amount should be greater than minimum validator staking amount
    require(amount >= _stakingConfig.getMinValidatorStakeAmount(), "too low");
    require(amount % BALANCE_COMPACT_PRECISION == 0, "no remainder");
    // transfer tokens
    require(_erc20Token.transferFrom(msg.sender, address(this), amount), "TokenStaking: failed to transfer");
    // add new validator as pending
    _addValidator(validatorAddress, msg.sender, ValidatorStatus.Pending, commissionRate, amount, nextEpoch());
}
```

Figure 3 The source code of *registervalidator* function(Fixed)

[Ankr-staking-2] AdvanceStakingRewards execution order issue

Severity Level	**Medium**
Type	Business Security
Lines	LiquidStakingPool.sol #L70-93

Description The advanceStakingRewards modifier is called when staking and withdrawing, there is a issue with the order of execution. First calculate the stakeableDust through *_calcUnclaimedDelegatorFee* and *calcAvailableForDelegateAmount*. At this time, the contract has not yet received the dust reward and calls *_delegateTo* to stake. then the function will not be called successfully.

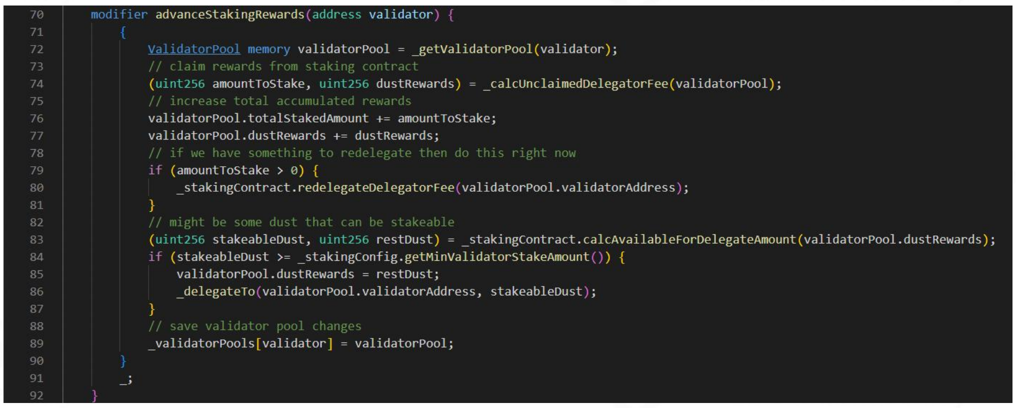


```
modifier advanceStakingRewards(address validator) {
    {
        ValidatorPool memory validatorPool = _getValidatorPool(validator);
        // claim rewards from staking contract
        (uint256 amountToStake, uint256 dustRewards) = _calcUnclaimedDelegatorFee(validatorPool);
        // increase total accumulated rewards
        validatorPool.totalStakedAmount += amountToStake;
        validatorPool.dustRewards += dustRewards;

        // might be some dust that can be stakeable
        (uint256 stakeableDust, uint256 restDust) = _stakingContract.calcAvailableForDelegateAmount(validatorPool.dustReward
        if (stakeableDust >= _stakingConfig.getMinValidatorStakeAmount()) {
            validatorPool.dustRewards = restDust;
            _delegateTo(validatorPool.validatorAddress, stakeableDust);
        }
        // save validator pool changes
        _validatorPools[validator] = validatorPool;
        // if we have something to redelegate then do this right now
        if (amountToStake > 0) {
            _stakingContract.redelegateDelegatorFee(validatorPool.validatorAddress);
        }
    }
    _;
}
```

Figure 4 The source code of advanceStakingRewards modifier

Recommendations It is recommended to change the calling order of the functions

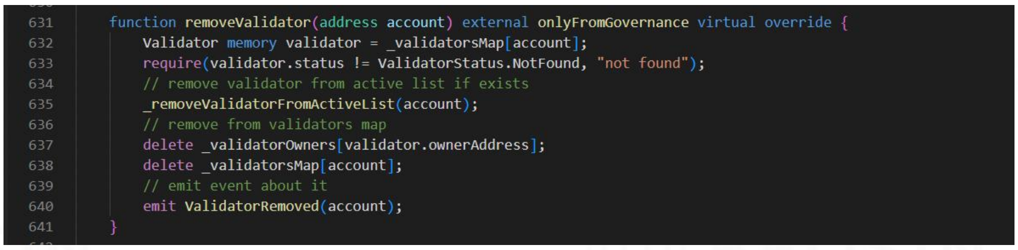
Status Fixed.



```
modifier advanceStakingRewards(address validator) {
    {
        ValidatorPool memory validatorPool = _getValidatorPool(validator);
        // claim rewards from staking contract
        (uint256 amountToStake, uint256 dustRewards) = _calcUnclaimedDelegatorFee(validatorPool);
        // increase total accumulated rewards
        validatorPool.totalStakedAmount += amountToStake;
        validatorPool.dustRewards += dustRewards;
        // if we have something to redelegate then do this right now
        if (amountToStake > 0) {
            _stakingContract.redelegateDelegatorFee(validatorPool.validatorAddress);
        }
        // might be some dust that can be stakeable
        (uint256 stakeableDust, uint256 restDust) = _stakingContract.calcAvailableForDelegateAmount(validatorPool.dustRewards);
        if (stakeableDust >= _stakingConfig.getMinValidatorStakeAmount()) {
            validatorPool.dustRewards = restDust;
            _delegateTo(validatorPool.validatorAddress, stakeableDust);
        }
        // save validator pool changes
        _validatorPools[validator] = validatorPool;
    }
    _;
}
```

Figure 5 The source code of advanceStakingRewards modifier(Fixed)

[Ankr-staking-3] Deleted validator user cannot extract

Severity Level	Medium
Type	Business Security
Lines	Staking.sol #L631-641
Description	When the Governance deletes the validator through the *removevalidator* function, the user will not be able to withdraw the staked principal through the *undelegate* function.  Figure 6 The source code of *removevalidator* function
Recommendations	It is recommended to add the extraction function for the deleted validator.
Status	Fixed. The project side deleted the *removevalidator* function.

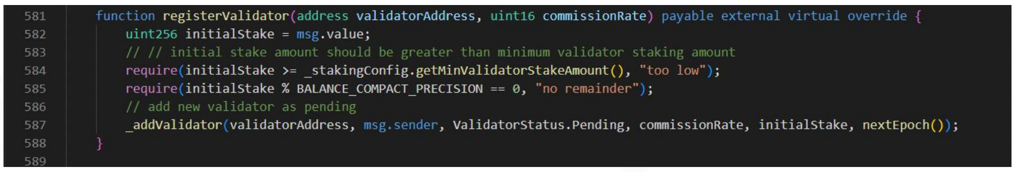
```
function removeValidator(address account) external onlyFromGovernance virtual override {
    Validator memory validator = _validatorsMap[account];
    require(validator.status != ValidatorStatus.NotFound, "not found");
    // remove validator from active list if exists
    _removeValidatorFromActiveList(account);
    // remove from validators map
    delete _validatorOwners[validator.ownerAddress];
    delete _validatorsMap[account];
    // emit event about it
    emit ValidatorRemoved(account);
}
```

[Ankr-staking-4] Asset types are not uniform

Severity Level	Medium
Type	Business Security
Lines	Staking.sol #L581-588

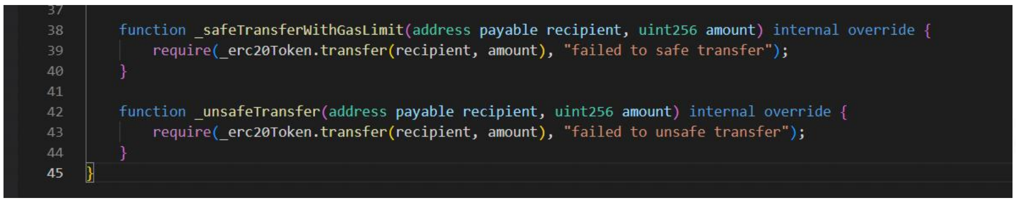
Description

When the user registers with the validator through *registervalidator* function, they use the BNB as the stake asset and create a corresponding number of stake records. However, the specified BEP20 tokens are used when receiving awards and withdrawing for settlement. Then when the registered user withdraws, the BEP20 tokens staked by other users will be withdrawn, and the BNB staked in the contract will not be able to be withdrawn.



```
581    function registerValidator(address validatorAddress, uint16 commissionRate) payable external virtual override {
582        uint256 initialStake = msg.value;
583        // // initial stake amount should be greater than minimum validator staking amount
584        require(initialStake >= _stakingConfig.getMinValidatorStakeAmount(), "too low");
585        require(initialStake % BALANCE_COMPACT_PRECISION == 0, "no remainder");
586        // add new validator as pending
587        _addValidator(validatorAddress, msg.sender, ValidatorStatus.Pending, commissionRate, initialStake, nextEpoch());
588    }
589
```

Figure 7 The source code of *registervalidator* function



```
    function _safeTransferWithGasLimit(address payable recipient, uint256 amount) internal override {
        require(_erc20Token.transfer(recipient, amount), "failed to safe transfer");
    }

    function _unsafeTransfer(address payable recipient, uint256 amount) internal override {
        require(_erc20Token.transfer(recipient, amount), "failed to unsafe transfer");
    }
}
```

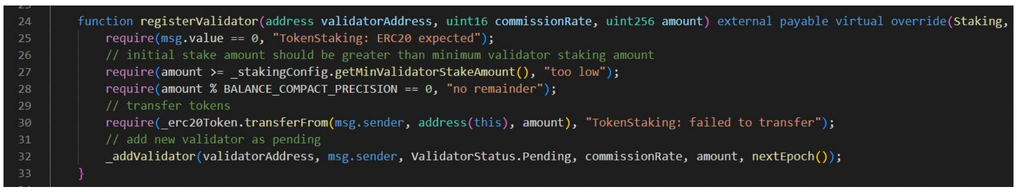
Figure 8 The source code of *transfer* functions

Recommendations

It is recommended to unify the asset types used in the contract.

Status

Fixed.



```
    function registerValidator(address validatorAddress, uint16 commissionRate, uint256 amount) external payable virtual override(Staking,
        require(msg.value == 0, "TokenStaking: ERC20 expected");
        // initial stake amount should be greater than minimum validator staking amount
        require(amount >= _stakingConfig.getMinValidatorStakeAmount(), "too low");
        require(amount % BALANCE_COMPACT_PRECISION == 0, "no remainder");
        // transfer tokens
        require(_erc20Token.transferFrom(msg.sender, address(this), amount), "TokenStaking: failed to transfer");
        // add new validator as pending
        _addValidator(validatorAddress, msg.sender, ValidatorStatus.Pending, commissionRate, amount, nextEpoch());
    }
```

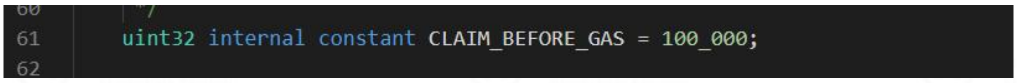
Figure 9 The source code of *registervalidator* function(Fixed)

[Ankr-staking-5] Delete staking records without rewards

Severity Level	Low
Type	Business Security
Lines	Staking.sol #L441-471

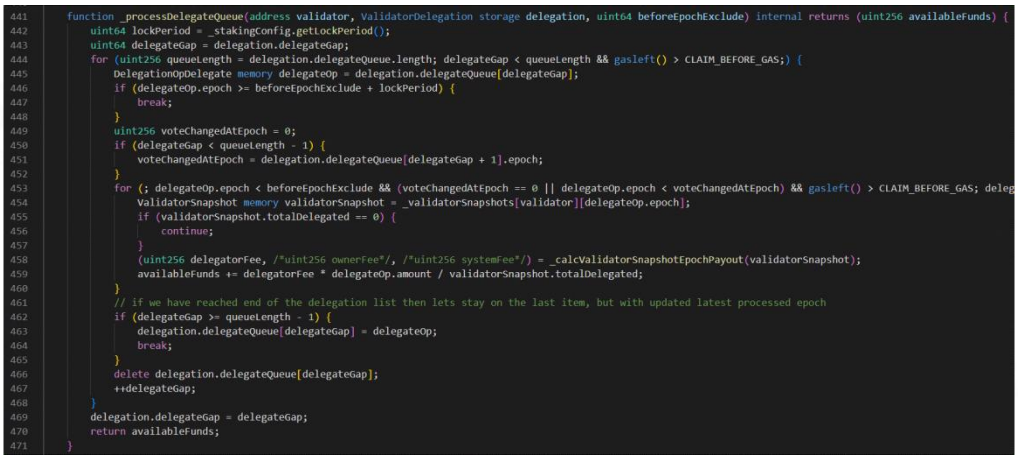
Description

When the user's gasleft meets the CLAIM_BEFORE_GAS of the first cycle, but not the CLAIM_BEFORE_GAS of the second cycle. There will be cases where users' staking records are deleted without rewards being issued.



```
    uint32 internal constant CLAIM_BEFORE_GAS = 100_000;
```

Figure 10 The source code of CLAIM_BEFORE_GAS



```
441  function _processDelegateQueue(address validator, ValidatorDelegation storage delegation, uint64 beforeEpochExclude) internal returns (uint256 availableFunds) {
442      uint64 lockPeriod = _stakingConfig.getLockPeriod();
443      uint64 delegateGap = delegation.delegateGap;
444      for (uint256 queueLength = delegation.delegateQueue.length; delegateGap < queueLength && gasleft() > CLAIM_BEFORE_GAS;) {
445          DelegationOpDelegate memory delegateOp = delegation.delegateQueue[delegateGap];
446          if (delegateOp.epoch >= beforeEpochExclude + lockPeriod) {
447              break;
448          }
449          uint256 voteChangedAtEpoch = 0;
450          if (delegateGap < queueLength - 1) {
451              voteChangedAtEpoch = delegation.delegateQueue[delegateGap + 1].epoch;
452          }
453          for (; delegateOp.epoch < beforeEpochExclude && (voteChangedAtEpoch == 0 || delegateOp.epoch < voteChangedAtEpoch) && gasleft() > CLAIM_BEFORE_GAS; deleg
454              ValidatorSnapshot memory validatorSnapshot = _validatorSnapshots[validator][delegateOp.epoch];
455              if (validatorSnapshot.totalDelegated == 0) {
456                  continue;
457              }
458              (uint256 delegatorFee, /*uint256 ownerFee*/, /*uint256 systemFee*/) = _calcValidatorSnapshotEpochPayout(validatorSnapshot);
459              availableFunds += delegatorFee * delegateOp.amount / validatorSnapshot.totalDelegated;
460          }
461          // if we have reached end of the delegation list then lets stay on the last item, but with updated latest processed epoch
462          if (delegateGap >= queueLength - 1) {
463              delegation.delegateQueue[delegateGap] = delegateOp;
464              break;
465          }
466          delete delegation.delegateQueue[delegateGap];
467          ++delegateGap;
468      }
469      delegation.delegateGap = delegateGap;
470      return availableFunds;
471  }
```

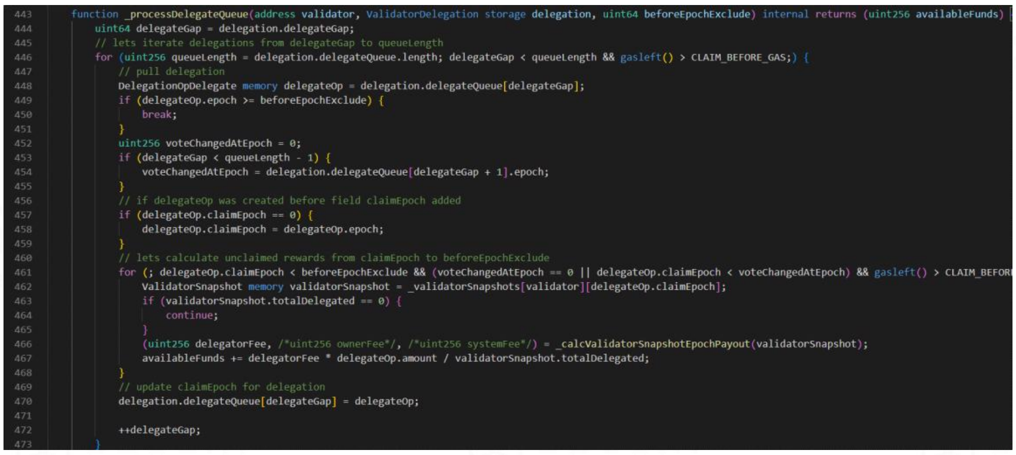
Figure 11 The source code of *_processDelegateQueue* function

Recommendations

It is suggested that the gasleft judgment of the second cycle is smaller than the first reasonable value.

Status

Fixed. The project party modified the collection logic.



```
function _processDelegateQueue(address validator, ValidatorDelegation storage delegation, uint64 beforeEpochExclude) internal returns (uint256 availableFunds) 
    uint64 delegateGap = delegation.delegateGap;
    // lets iterate delegations from delegateGap to queueLength
    for (uint256 queueLength = delegation.delegateQueue.length; delegateGap < queueLength && gasleft() > CLAIM_BEFORE_GAS;) {
        // pull delegation
        DelegationOpDelegate memory delegateOp = delegation.delegateQueue[delegateGap];
        if (delegateOp.epoch >= beforeEpochExclude) {
            break;
        }
        uint256 voteChangedAtEpoch = 0;
        if (delegateGap < queueLength - 1) {
            voteChangedAtEpoch = delegation.delegateQueue[delegateGap + 1].epoch;
        }
        // if delegateOp was created before field claimEpoch added
        if (delegateOp.claimEpoch == 0) {
            delegateOp.claimEpoch = delegateOp.epoch;
        }
        // lets calculate unclaimed rewards from claimEpoch to beforeEpochExclude
        for (; delegateOp.claimEpoch < beforeEpochExclude && (voteChangedAtEpoch == 0 || delegateOp.claimEpoch < voteChangedAtEpoch) && gasleft() > CLAIM_BEFOR
            ValidatorSnapshot memory validatorSnapshot = _validatorSnapshots[validator][delegateOp.claimEpoch];
            if (validatorSnapshot.totalDelegated == 0) {
                continue;
            }
            (uint256 delegatorFee, /*uint256 ownerFee*/, /*uint256 systemFee*/) = _calcValidatorSnapshotEpochPayout(validatorSnapshot);
            availableFunds += delegatorFee * delegateOp.amount / validatorSnapshot.totalDelegated;
        }
        // update claimEpoch for delegation
        delegation.delegateQueue[delegateGap] = delegateOp;

        ++delegateGap;
    }
```

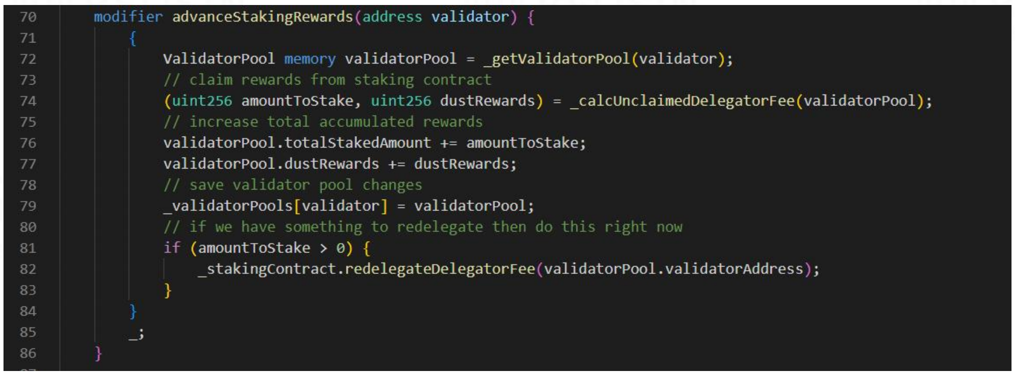
Figure 12 The source code of *_processDelegateQueue* function(Fixed)

[Ankr-staking-6] The dustRewards reward cannot be withdrawn

Severity Level	Low
Type	Business Security
Lines	LiquidStakingPool.sol #L70-86

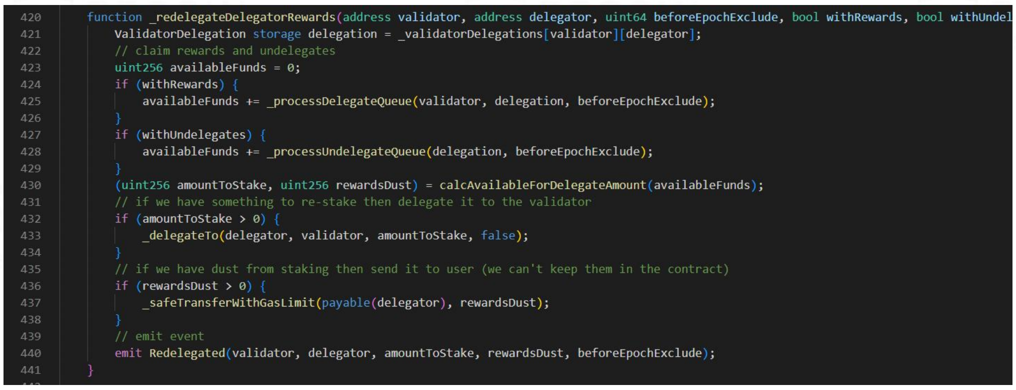
Description

When the advanceStakingRewards modifier calls the *redelegateDelegatorFee* function, the rewardsDust transferred to the LiquidStakingPool contract is not processed, resulting in rewardsDust being locked in the contract and unable to be withdrawn.



```
70      modifier advanceStakingRewards(address validator) {
71          {
72              ValidatorPool memory validatorPool = _getValidatorPool(validator);
73              // claim rewards from staking contract
74              (uint256 amountToStake, uint256 dustRewards) = _calcUnclaimedDelegatorFee(validatorPool);
75              // increase total accumulated rewards
76              validatorPool.totalStakedAmount += amountToStake;
77              validatorPool.dustRewards += dustRewards;
78              // save validator pool changes
79              _validatorPools[validator] = validatorPool;
80              // if we have something to redelegate then do this right now
81              if (amountToStake > 0) {
82                  _stakingContract.redelegateDelegatorFee(validatorPool.validatorAddress);
83              }
84          }
85          _;
86      }
```

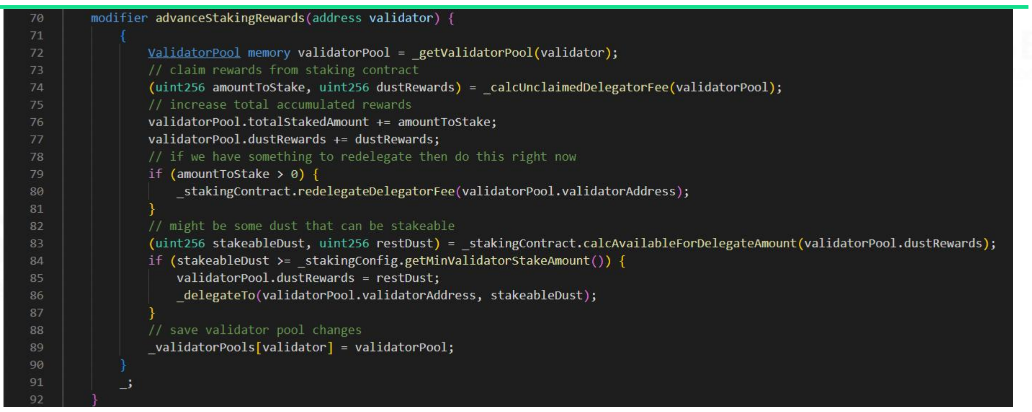
Figure 13 The source code of advanceStakingRewards modifier



```
function _redelegateDelegatorRewards(address validator, address delegator, uint64 beforeEpochExclude, bool withRewards, bool withUndel
    ValidatorDelegation storage delegation = _validatorDelegations[validator][delegator];
    // claim rewards and undelegates
    uint256 availableFunds = 0;
    if (withRewards) {
        availableFunds += _processDelegateQueue(validator, delegation, beforeEpochExclude);
    }
    if (withUndelegates) {
        availableFunds += _processUndelegateQueue(delegation, beforeEpochExclude);
    }
    (uint256 amountToStake, uint256 rewardsDust) = calcAvailableForDelegateAmount(availableFunds);
    // if we have something to re-stake then delegate it to the validator
    if (amountToStake > 0) {
        _delegateTo(delegator, validator, amountToStake, false);
    }
    // if we have dust from staking then send it to user (we can't keep them in the contract)
    if (rewardsDust > 0) {
        _safeTransferWithGasLimit(payable(delegator), rewardsDust);
    }
    // emit event
    emit Redelegated(validator, delegator, amountToStake, rewardsDust, beforeEpochExclude);
}
```

Figure 14 The source code of _redelegateDelegatorRewards function

Recommendations	It is recommended to increase the extraction function of rewardsDust.
Status	Fixed.

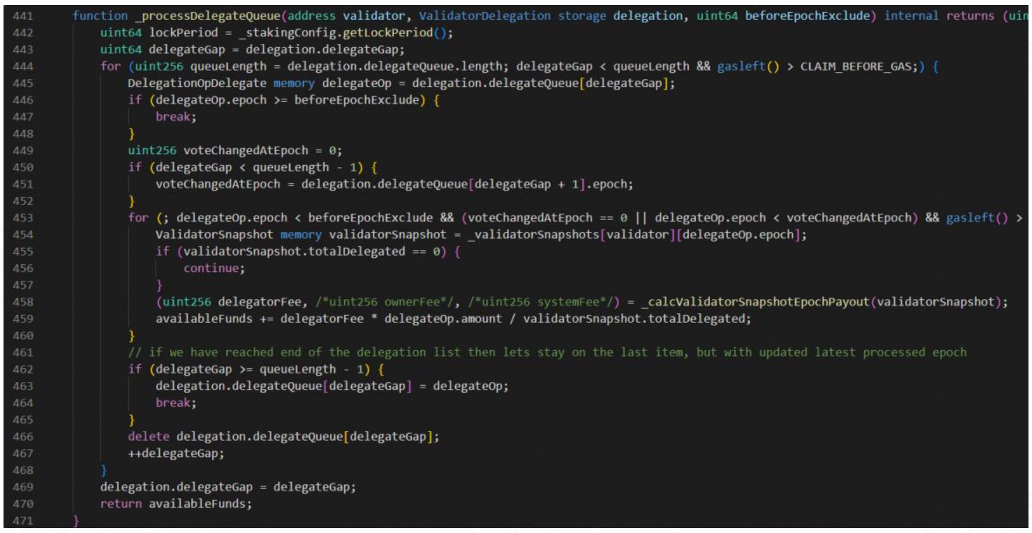


```
modifier advanceStakingRewards(address validator) {
    {
        ValidatorPool memory validatorPool = _getValidatorPool(validator);
        // claim rewards from staking contract
        (uint256 amountToStake, uint256 dustRewards) = _calcUnclaimedDelegatorFee(validatorPool);
        // increase total accumulated rewards
        validatorPool.totalStakedAmount += amountToStake;
        validatorPool.dustRewards += dustRewards;
        // if we have something to redelegate then do this right now
        if (amountToStake > 0) {
            _stakingContract.redelegateDelegatorFee(validatorPool.validatorAddress);
        }
        // might be some dust that can be stakeable
        (uint256 stakeableDust, uint256 restDust) = _stakingContract.calcAvailableForDelegateAmount(validatorPool.dustRewards);
        if (stakeableDust >= _stakingConfig.getMinValidatorStakeAmount()) {
            validatorPool.dustRewards = restDust;
            _delegateTo(validatorPool.validatorAddress, stakeableDust);
        }
        // save validator pool changes
        _validatorPools[validator] = validatorPool;
    }
    _;
}
```

Figure 15 The source code of advanceStakingRewards modifier(Fixed)

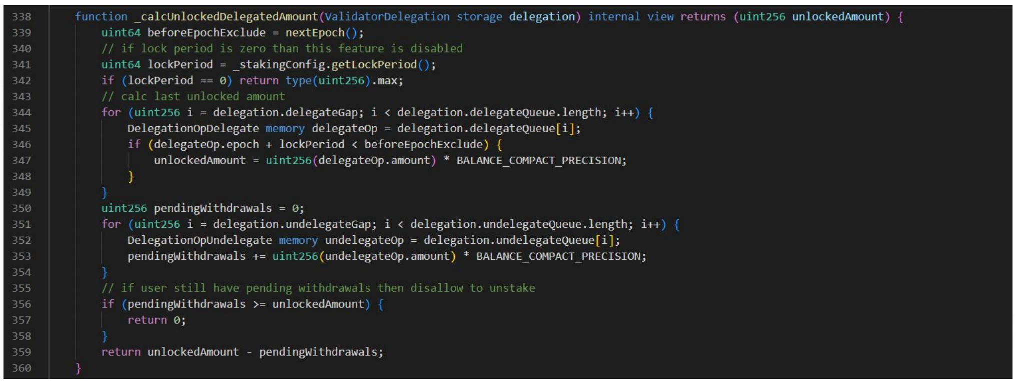
[Ankr-staking-7] Inaccurate principal withdrawal amount

Severity Level	Low
Type	Business Security
Lines	Staking.sol #L441-471, L338-360
Description	The sequence of receiving prizes and principal withdrawals will affect the next principal withdrawal amount. If the reward is claimed first, the next principal withdrawal amount will be reduced.



```
441     function _processDelegateQueue(address validator, ValidatorDelegation storage delegation, uint64 beforeEpochExclude) internal returns (uin
442         uint64 lockPeriod = _stakingConfig.getLockPeriod();
443         uint64 delegateGap = delegation.delegateGap;
444         for (uint256 queueLength = delegation.delegateQueue.length; delegateGap < queueLength && gasleft() > CLAIM_BEFORE_GAS;) {
445             DelegationOpDelegate memory delegateOp = delegation.delegateQueue[delegateGap];
446             if (delegateOp.epoch >= beforeEpochExclude) {
447                 break;
448             }
449             uint256 voteChangedAtEpoch = 0;
450             if (delegateGap < queueLength - 1) {
451                 voteChangedAtEpoch = delegation.delegateQueue[delegateGap + 1].epoch;
452             }
453             for (; delegateOp.epoch < beforeEpochExclude && (voteChangedAtEpoch == 0 || delegateOp.epoch < voteChangedAtEpoch) && gasleft() >
454                 ValidatorSnapshot memory validatorSnapshot = _validatorSnapshots[validator][delegateOp.epoch];
455                 if (validatorSnapshot.totalDelegated == 0) {
456                     continue;
457                 }
458                 (uint256 delegatorFee, /*uint256 ownerFee*/, /*uint256 systemFee*/) = _calcValidatorSnapshotEpochPayout(validatorSnapshot);
459                 availableFunds += delegatorFee * delegateOp.amount / validatorSnapshot.totalDelegated;
460             }
461             // if we have reached end of the delegation list then lets stay on the last item, but with updated latest processed epoch
462             if (delegateGap >= queueLength - 1) {
463                 delegation.delegateQueue[delegateGap] = delegateOp;
464                 break;
465             }
466             delete delegation.delegateQueue[delegateGap];
467             ++delegateGap;
468         }
469         delegation.delegateGap = delegateGap;
470         return availableFunds;
471     }
```

Figure 16 The source code of *_processDelegateQueue* function



```
338     function _calcUnlockedDelegatedAmount(ValidatorDelegation storage delegation) internal view returns (uint256 unlockedAmount) {
339         uint64 beforeEpochExclude = nextEpoch();
340         // if lock period is zero than this feature is disabled
341         uint64 lockPeriod = _stakingConfig.getLockPeriod();
342         if (lockPeriod == 0) return type(uint256).max;
343         // calc last unlocked amount
344         for (uint256 i = delegation.delegateGap; i < delegation.delegateQueue.length; i++) {
345             DelegationOpDelegate memory delegateOp = delegation.delegateQueue[i];
346             if (delegateOp.epoch + lockPeriod < beforeEpochExclude) {
347                 unlockedAmount = uint256(delegateOp.amount) * BALANCE_COMPACT_PRECISION;
348             }
349         }
350         uint256 pendingWithdrawals = 0;
351         for (uint256 i = delegation.undelegateGap; i < delegation.undelegateQueue.length; i++) {
352             DelegationOpUndelegate memory undelegateOp = delegation.undelegateQueue[i];
353             pendingWithdrawals += uint256(undelegateOp.amount) * BALANCE_COMPACT_PRECISION;
354         }
355         // if user still have pending withdrawals then disallow to unstake
356         if (pendingWithdrawals >= unlockedAmount) {
357             return 0;
358         }
359         return unlockedAmount - pendingWithdrawals;
360     }
```

Figure 17 The source code of *_calcUnlockedDelegatedAmount* function

Recommendations	It is recommended to keep stake records.
Status	Fixed.

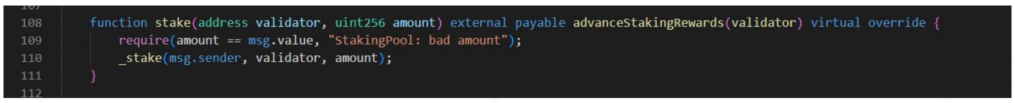
[Ankr-staking-8] Unreasonable *stake* function

Severity Level Low

Type Business Security

Lines LiquidStakingPool.sol #L108-111

Description The LiquidStakingPool contract has redundant *stake* functions, and the staked assets are platform tokens. TokenLiquidStakingPool will inherit this function and the call will fail.

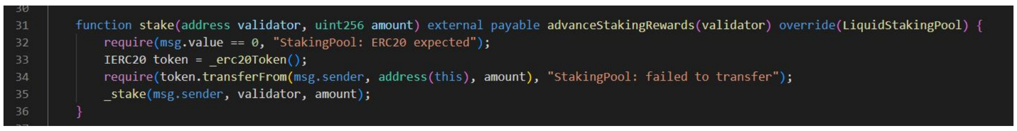


```
108     function stake(address validator, uint256 amount) external payable advanceStakingRewards(validator) virtual override {
109         require(amount == msg.value, "StakingPool: bad amount");
110         _stake(msg.sender, validator, amount);
111     }
```

Figure 18 The source code of *stake* function

Recommendations It is recommended to remove this function.

Status Fixed.



```
    function stake(address validator, uint256 amount) external payable advanceStakingRewards(validator) override(LiquidStakingPool) {
        require(msg.value == 0, "StakingPool: ERC20 expected");
        IERC20 token = _erc20Token();
        require(token.transferFrom(msg.sender, address(this), amount), "StakingPool: failed to transfer");
        _stake(msg.sender, validator, amount);
    }
```

Figure 19 The source code of *stake* function(Fixed)

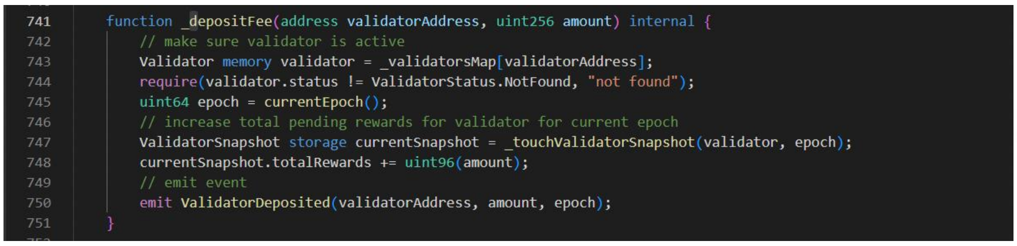
[Ankr-staking-9] Not updating change.at to global variable

Severity Level Info

Type Business Security

Lines Staking.sol #L741-751

Description After validator updates change.at as memory, it does not assign a value to the global variable. Causes the recorded data to be incorrect.

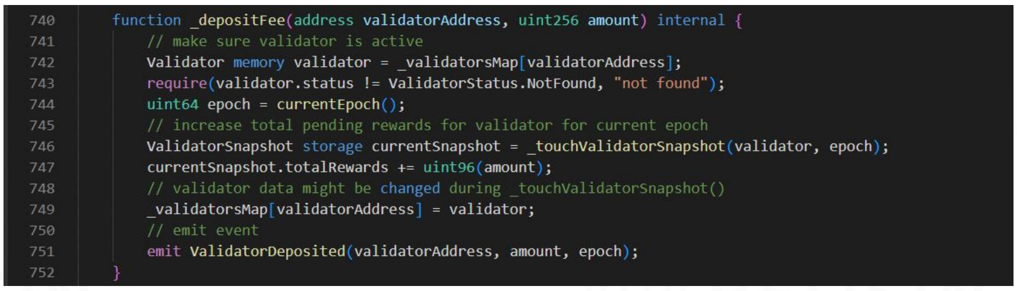


```
741    function _depositFee(address validatorAddress, uint256 amount) internal {
742        // make sure validator is active
743        Validator memory validator = _validatorsMap[validatorAddress];
744        require(validator.status != ValidatorStatus.NotFound, "not found");
745        uint64 epoch = currentEpoch();
746        // increase total pending rewards for validator for current epoch
747        ValidatorSnapshot storage currentSnapshot = _touchValidatorSnapshot(validator, epoch);
748        currentSnapshot.totalRewards += uint96(amount);
749        // emit event
750        emit ValidatorDeposited(validatorAddress, amount, epoch);
751    }
```

Figure 20 The source code of *_depositFee* function

Recommendations It is recommended to update global variables.

Status Fixed.

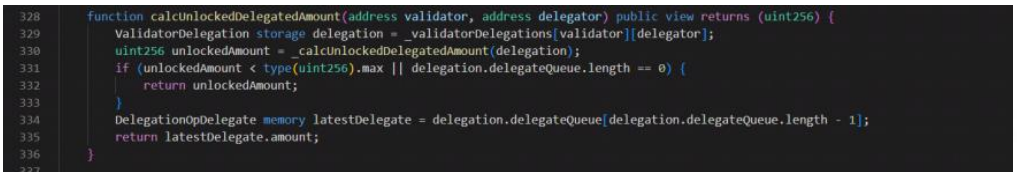


```
740    function _depositFee(address validatorAddress, uint256 amount) internal {
741        // make sure validator is active
742        Validator memory validator = _validatorsMap[validatorAddress];
743        require(validator.status != ValidatorStatus.NotFound, "not found");
744        uint64 epoch = currentEpoch();
745        // increase total pending rewards for validator for current epoch
746        ValidatorSnapshot storage currentSnapshot = _touchValidatorSnapshot(validator, epoch);
747        currentSnapshot.totalRewards += uint96(amount);
748        // validator data might be changed during _touchValidatorSnapshot()
749        _validatorsMap[validatorAddress] = validator;
750        // emit event
751        emit ValidatorDeposited(validatorAddress, amount, epoch);
752    }
```

Figure 21 The source code of *_depositFee* function(Fixed)

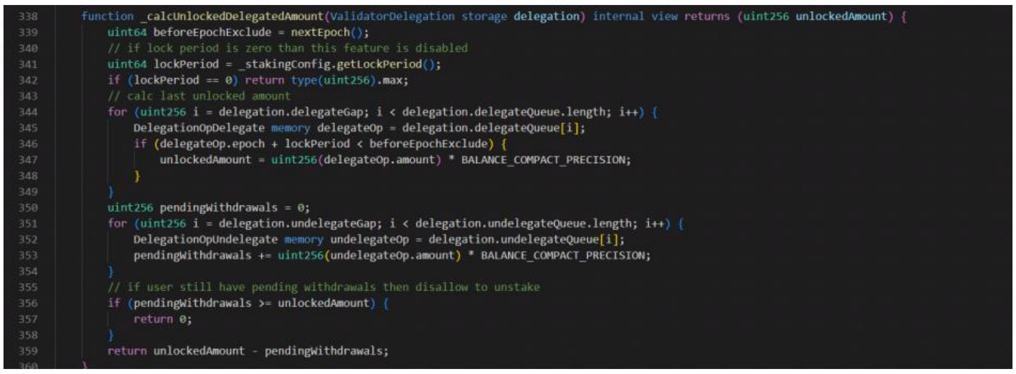
[Ankr-staking-10] Inconsistent accuracy

Severity Level	Info
Type	Business Security
Lines	Staking.sol #L328-360
Description	The latestDelegate used by the *calcUnlockedDelegatedAmount* function returned without restoring precision, resulting in a reduced drawable amount for the user.



```
function calcUnlockedDelegatedAmount(address validator, address delegator) public view returns (uint256) {
    ValidatorDelegation storage delegation = _validatorDelegations[validator][delegator];
    uint256 unlockedAmount = _calcUnlockedDelegatedAmount(delegation);
    if (unlockedAmount < type(uint256).max || delegation.delegateQueue.length == 0) {
        return unlockedAmount;
    }
    DelegationOpDelegate memory latestDelegate = delegation.delegateQueue[delegation.delegateQueue.length - 1];
    return latestDelegate.amount;
}
```

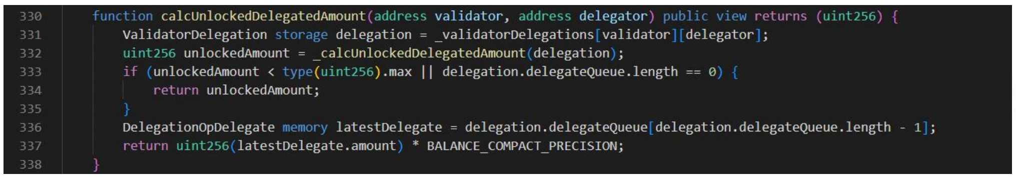
Figure 22 The source code of *calcUnlockedDelegatedAmount* function



```
function _calcUnlockedDelegatedAmount(ValidatorDelegation storage delegation) internal view returns (uint256 unlockedAmount) {
    uint64 beforeEpochExclude = nextEpoch();
    // if lock period is zero than this feature is disabled
    uint64 lockPeriod = _stakingConfig.getLockPeriod();
    if (lockPeriod == 0) return type(uint256).max;
    // calc last unlocked amount
    for (uint256 i = delegation.delegateGap; i < delegation.delegateQueue.length; i++) {
        DelegationOpDelegate memory delegateOp = delegation.delegateQueue[i];
        if (delegateOp.epoch + lockPeriod < beforeEpochExclude) {
            unlockedAmount = uint256(delegateOp.amount) * BALANCE_COMPACT_PRECISION;
        }
    }
    uint256 pendingWithdrawals = 0;
    for (uint256 i = delegation.undelegateGap; i < delegation.undelegateQueue.length; i++) {
        DelegationOpUndelegate memory undelegateOp = delegation.undelegateQueue[i];
        pendingWithdrawals += uint256(undelegateOp.amount) * BALANCE_COMPACT_PRECISION;
    }
    // if user still have pending withdrawals then disallow to unstake
    if (pendingWithdrawals >= unlockedAmount) {
        return 0;
    }
    return unlockedAmount - pendingWithdrawals;
}
```

Figure 23 The source code of *_calcUnlockedDelegatedAmount* function

Recommendations	It is recommended to restore the accuracy to the same.
Status	Fixed.



```
function calcUnlockedDelegatedAmount(address validator, address delegator) public view returns (uint256) {
    ValidatorDelegation storage delegation = _validatorDelegations[validator][delegator];
    uint256 unlockedAmount = _calcUnlockedDelegatedAmount(delegation);
    if (unlockedAmount < type(uint256).max || delegation.delegateQueue.length == 0) {
        return unlockedAmount;
    }
    DelegationOpDelegate memory latestDelegate = delegation.delegateQueue[delegation.delegateQueue.length - 1];
    return uint256(latestDelegate.amount) * BALANCE_COMPACT_PRECISION;
}
```

Figure 24 The source code of *calcUnlockedDelegatedAmount* function(Fixed)

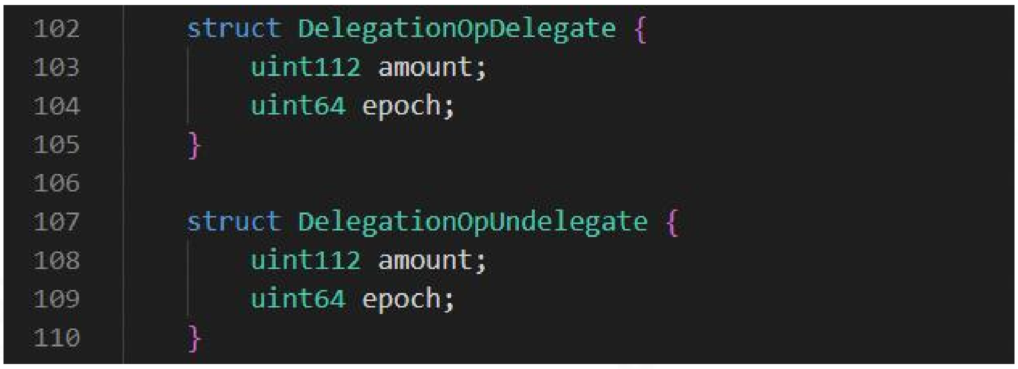
[Ankr-staking-11] Variable types are inconsistent

Severity Level Info

Type Business Security

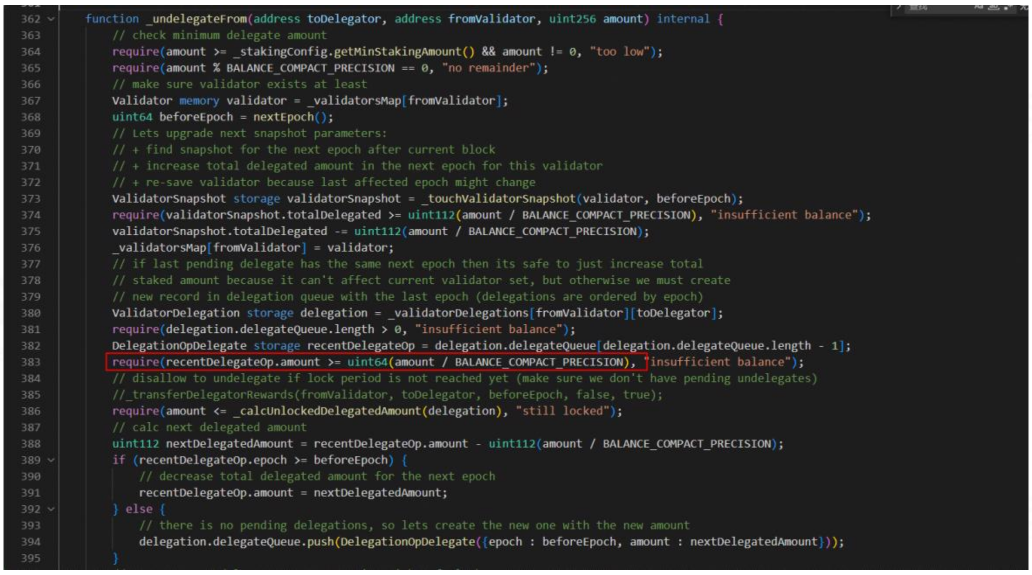
Lines Staking.sol #L102-110

Description The uint64 type used by the *_undelegateFrom* function is inconsistent with the definition.



```
102     struct DelegationOpDelegate {
103         uint112 amount;
104         uint64 epoch;
105     }
106
107     struct DelegationOpUndelegate {
108         uint112 amount;
109         uint64 epoch;
110     }
```

Figure 25 The source code of DelegationOp

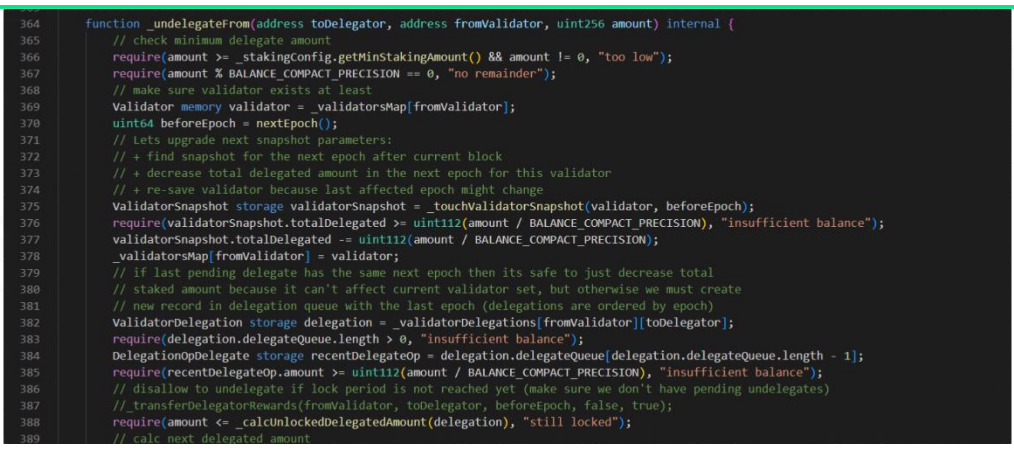


```
    function _undelegateFrom(address toDelegator, address fromValidator, uint256 amount) internal {
        // check minimum delegate amount
        require(amount >= _stakingConfig.getMinStakingAmount() && amount != 0, "too low");
        require(amount % BALANCE_COMPACT_PRECISION == 0, "no remainder");
        // make sure validator exists at least
        Validator memory validator = _validatorsMap[fromValidator];
        uint64 beforeEpoch = nextEpoch();
        // Lets upgrade next snapshot parameters:
        // + find snapshot for the next epoch after current block
        // + increase total delegated amount in the next epoch for this validator
        // + re-save validator because last affected epoch might change
        ValidatorSnapshot storage validatorSnapshot = _touchValidatorSnapshot(validator, beforeEpoch);
        require(validatorSnapshot.totalDelegated >= uint112(amount / BALANCE_COMPACT_PRECISION), "insufficient balance");
        validatorSnapshot.totalDelegated -= uint112(amount / BALANCE_COMPACT_PRECISION);
        _validatorsMap[fromValidator] = validator;
        // if last pending delegate has the same next epoch then its safe to just increase total
        // staked amount because it can't affect current validator set, but otherwise we must create
        // new record in delegation queue with the last epoch (delegations are ordered by epoch)
        ValidatorDelegation storage delegation = _validatorDelegations[fromValidator][toDelegator];
        require(delegation.delegateQueue.length > 0, "insufficient balance");
        DelegationOpDelegate storage recentDelegateOp = delegation.delegateQueue[delegation.delegateQueue.length - 1];
        require(recentDelegateOp.amount >= uint64(amount / BALANCE_COMPACT_PRECISION), "insufficient balance");
        // disallow to undelegate if lock period is not reached yet (make sure we don't have pending undelegates)
        //_transferDelegatorRewards(fromValidator, toDelegator, beforeEpoch, false, true);
        require(amount <= _calcUnlockedDelegatedAmount(delegation), "still locked");
        // calc next delegated amount
        uint112 nextDelegatedAmount = recentDelegateOp.amount - uint112(amount / BALANCE_COMPACT_PRECISION);
        if (recentDelegateOp.epoch >= beforeEpoch) {
            // decrease total delegated amount for the next epoch
            recentDelegateOp.amount = nextDelegatedAmount;
        } else {
            // there is no pending delegations, so lets create the new one with the new amount
            delegation.delegateQueue.push(DelegationOpDelegate({epoch : beforeEpoch, amount : nextDelegatedAmount}));
        }
```

Figure 26 The source code of *_undelegateFrom* function

Recommendations It is recommended to unify the uint112 type.

Status Fixed.



```
function _undelegateFrom(address toDelegator, address fromValidator, uint256 amount) internal {
    // check minimum delegate amount
    require(amount >= _stakingConfig.getMinStakingAmount() && amount != 0, "too low");
    require(amount % BALANCE_COMPACT_PRECISION == 0, "no remainder");
    // make sure validator exists at least
    Validator memory validator = _validatorsMap[fromValidator];
    uint64 beforeEpoch = nextEpoch();
    // Lets upgrade next snapshot parameters:
    // + find snapshot for the next epoch after current block
    // + decrease total delegated amount in the next epoch for this validator
    // + re-save validator because last affected epoch might change
    ValidatorSnapshot storage validatorSnapshot = _touchValidatorSnapshot(validator, beforeEpoch);
    require(validatorSnapshot.totalDelegated >= uint112(amount / BALANCE_COMPACT_PRECISION), "insufficient balance");
    validatorSnapshot.totalDelegated -= uint112(amount / BALANCE_COMPACT_PRECISION);
    _validatorsMap[fromValidator] = validator;
    // if last pending delegate has the same next epoch then its safe to just decrease total
    // staked amount because it can't affect current validator set, but otherwise we must create
    // new record in delegation queue with the last epoch (delegations are ordered by epoch)
    ValidatorDelegation storage delegation = _validatorDelegations[fromValidator][toDelegator];
    require(delegation.delegateQueue.length > 0, "insufficient balance");
    DelegationOpDelegate storage recentDelegateOp = delegation.delegateQueue[delegation.delegateQueue.length - 1];
    require(recentDelegateOp.amount >= uint112(amount / BALANCE_COMPACT_PRECISION), "insufficient balance");
    // disallow to undelegate if lock period is not reached yet (make sure we don't have pending undelegates)
    //_transferDelegatorRewards(fromValidator, toDelegator, beforeEpoch, false, true);
    require(amount <= _calcUnlockedDelegatedAmount(delegation), "still locked");
    // calc next delegated amount
```

Figure 27 The source code of *_undelegateFrom* function(Fixed)

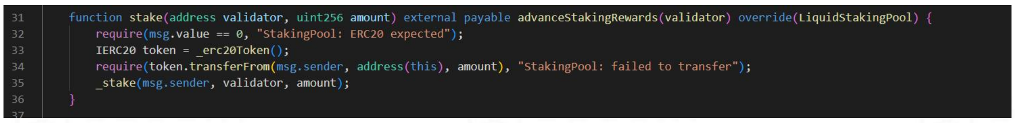
[Ankr-staking-12] Meaningless payable keyword

Severity Level Info

Type Business Security

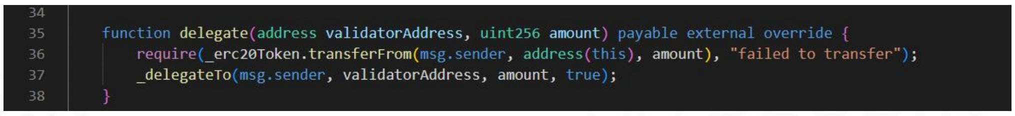
Lines TokenLiquidStakingPool.sol #L31-37 TokenStaking.sol #L35-38

Description The function has a payable type, and the user may mistakenly lock the BNB in the contract.



```
31    function stake(address validator, uint256 amount) external payable advanceStakingRewards(validator) override(LiquidStakingPool) {
32        require(msg.value == 0, "StakingPool: ERC20 expected");
33        IERC20 token = _erc20Token();
34        require(token.transferFrom(msg.sender, address(this), amount), "StakingPool: failed to transfer");
35        _stake(msg.sender, validator, amount);
36    }
37
```

Figure 28 The source code of *stake* function

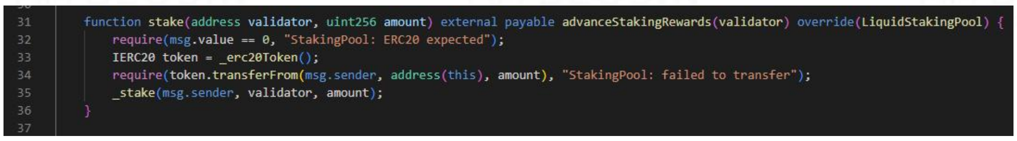


```
34
35    function delegate(address validatorAddress, uint256 amount) payable external override {
36        require(_erc20Token.transferFrom(msg.sender, address(this), amount), "failed to transfer");
37        _delegateTo(msg.sender, validatorAddress, amount, true);
38    }
```

Figure 29 The source code of *delegate* function

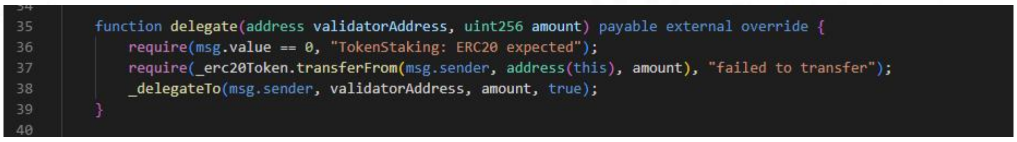
Recommendations It is recommended to delete the payable type.

Status Fixed.



```
31    function stake(address validator, uint256 amount) external payable advanceStakingRewards(validator) override(LiquidStakingPool) {
32        require(msg.value == 0, "StakingPool: ERC20 expected");
33        IERC20 token = _erc20Token();
34        require(token.transferFrom(msg.sender, address(this), amount), "StakingPool: failed to transfer");
35        _stake(msg.sender, validator, amount);
36    }
37
```

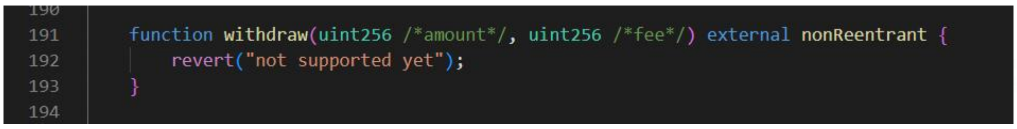
Figure 30 The source code of *stake* function(Fixed)



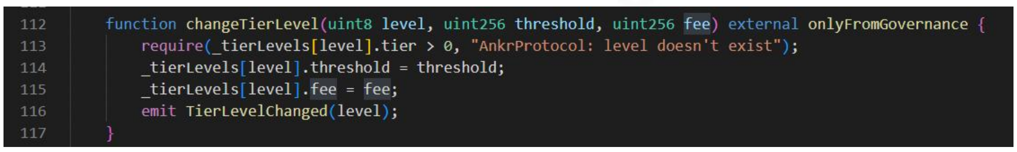
```
35    function delegate(address validatorAddress, uint256 amount) payable external override {
36        require(msg.value == 0, "TokenStaking: ERC20 expected");
37        require(_erc20Token.transferFrom(msg.sender, address(this), amount), "failed to transfer");
38        _delegateTo(msg.sender, validatorAddress, amount, true);
39    }
40
```

Figure 31 The source code of *delegate* function(Fixed)

[AnkrProtocol-1] Missing asset extraction interface

Severity Level	Low
Type	Business Security
Lines	AnkrProtocol.sol #L191-194
Description	The function of user extraction is not implemented in the contract, which causes the user to stake in the contract. The staked tokens are locked in the contract. And the fee charged cannot be withdrawn.  `function withdraw(uint256 /*amount*/, uint256 /*fee*/) external nonReentrant {` `revert("not supported yet");` `}` Figure 32 The source code of *withdraw* function
Recommendations	It is recommended to implement the extraction function.
Status	Acknowledged. According to the description of the project party, withdrawals are not possible for this smart contract. The project party added additional function called *transferCollectedFee* that allow to transfer locked funds to special contract that do fee distribution.

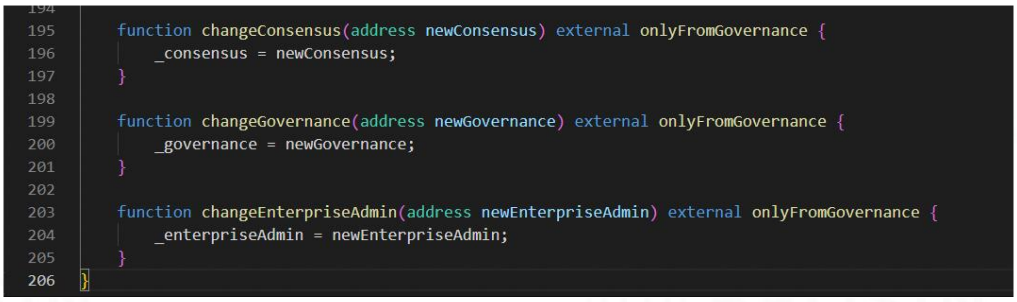
[AnkrProtocol-2] Stake level setting issue

Severity Level	Low
Type	Business Security
Lines	AnkrProtocol.sol #L112-117
Description	Governance has the authority to call the *changeTierLevel* function to arbitrarily change the threshold (staking level judgment amount) and fee (staking fee) of different levels. In the *createTierLevel* function, it is stipulated that only the larger the threshold, the higher the corresponding stake level, so that the threshold arbitrarily changed in the *changeTierLevel* function may not match the corresponding level. When Governance calls the *changeTierLevel* function to increase the threshold of the user's level, then when the user selects a stake, the level queried by the _*matchTierLevelOf* function will decrease. If there is no corresponding processing in _*lockDeposit*, it will lead to a loss of user level reduction.  Figure 33 The source code of *changeTierLevel* function
Recommendations	It is recommended to check whether the value is within the threshold range of the front and back levels when setting a new threshold in the *changeTierLevel* function.
Status	Acknowledged. According to the description of the project party, governance processes are managed and audited, the project party is not planning to change existing tier plans or add new tier plans.

```
    function changeTierLevel(uint8 level, uint256 threshold, uint256 fee) external onlyFromGovernance {
        require(_tierLevels[level].tier > 0, "AnkrProtocol: level doesn't exist");
        _tierLevels[level].threshold = threshold;
        _tierLevels[level].fee = fee;
        emit TierLevelChanged(level);
    }
```

[AnkrProtocol-3] Missing check for zero address

Severity Level	Low
Type	Business Security
Lines	AnkrProtocol.sol #L195-206
Description	When the following functions are called with corresponding permissions, there is a risk of transferring permissions to address zero.

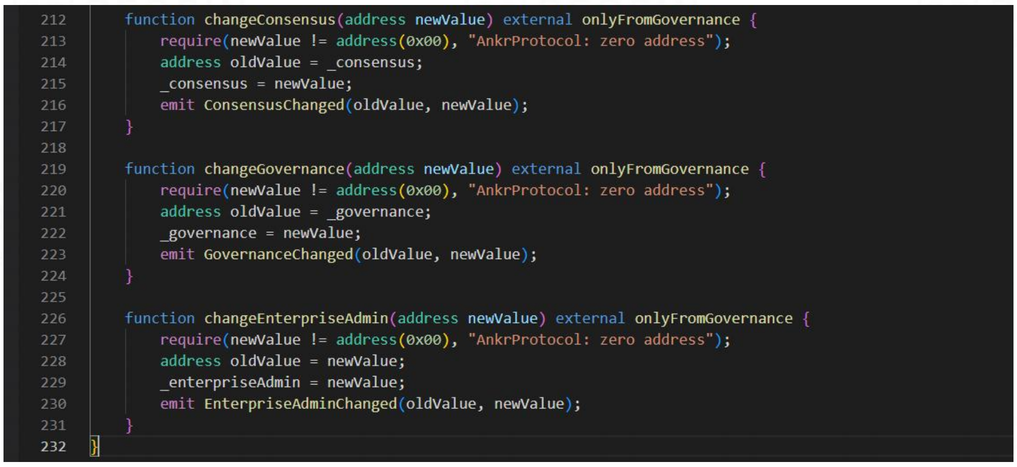


```
    function changeConsensus(address newConsensus) external onlyFromGovernance {
        _consensus = newConsensus;
    }

    function changeGovernance(address newGovernance) external onlyFromGovernance {
        _governance = newGovernance;
    }

    function changeEnterpriseAdmin(address newEnterpriseAdmin) external onlyFromGovernance {
        _enterpriseAdmin = newEnterpriseAdmin;
    }
}
```

Figure 34 The source code of unchecked functions

Recommendations It is recommended to add 0 address check.

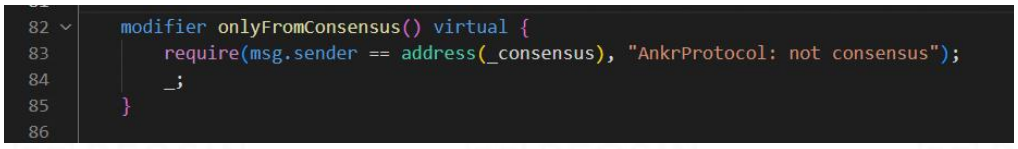
Status Fixed.



```
    function changeConsensus(address newValue) external onlyFromGovernance {
        require(newValue != address(0x00), "AnkrProtocol: zero address");
        address oldValue = _consensus;
        _consensus = newValue;
        emit ConsensusChanged(oldValue, newValue);
    }

    function changeGovernance(address newValue) external onlyFromGovernance {
        require(newValue != address(0x00), "AnkrProtocol: zero address");
        address oldValue = _governance;
        _governance = newValue;
        emit GovernanceChanged(oldValue, newValue);
    }

    function changeEnterpriseAdmin(address newValue) external onlyFromGovernance {
        require(newValue != address(0x00), "AnkrProtocol: zero address");
        address oldValue = newValue;
        _enterpriseAdmin = newValue;
        emit EnterpriseAdminChanged(oldValue, newValue);
    }
}
```

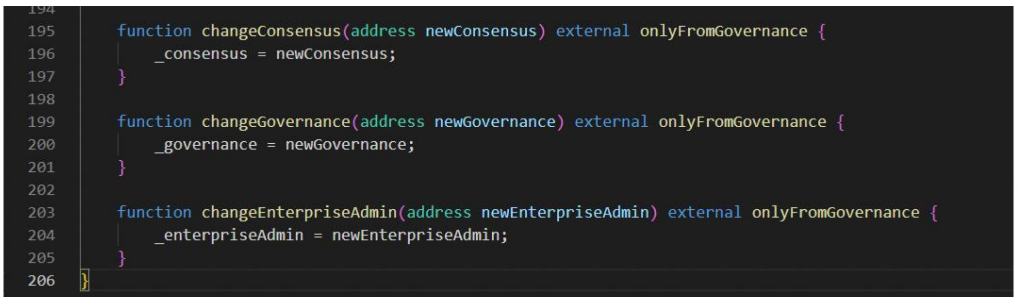
Figure 35 The source code of unchecked functions(Fixed)

[AnkrProtocol-4] Code redundancy

Severity Level	**Low**
Type	Business Security
Lines	AnkrProtocol.sol #L195-206
Description	The permission of consensus is not reflected in the contract.  ```solidity modifier onlyFromConsensus() virtual { require(msg.sender == address(_consensus), "AnkrProtocol: not consensus"); _; } ``` Figure 36 The source code of onlyFromGovernance modifier
Recommendations	It is recommended to remove this modifier.
Status	Fixed.

[AnkrProtocol-5] Lack of event triggering

Severity Level	Info
Type	Business Security
Lines	AnkrProtocol.sol #L195-206
Description	The following functions are missing event triggers.

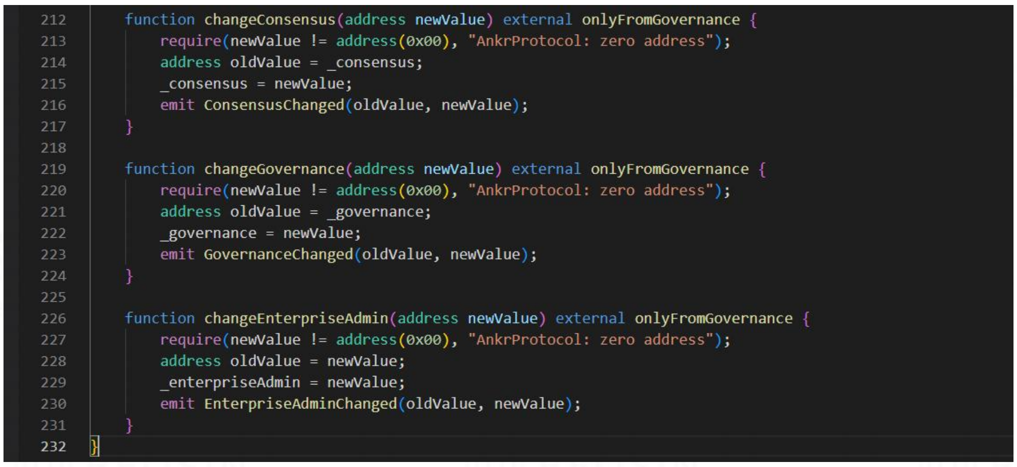


```
    function changeConsensus(address newConsensus) external onlyFromGovernance {
        _consensus = newConsensus;
    }

    function changeGovernance(address newGovernance) external onlyFromGovernance {
        _governance = newGovernance;
    }

    function changeEnterpriseAdmin(address newEnterpriseAdmin) external onlyFromGovernance {
        _enterpriseAdmin = newEnterpriseAdmin;
    }
}
```

Figure 37 The source code of untouched functions

Recommendations It is recommended to remove this modifier.

Status Fixed.

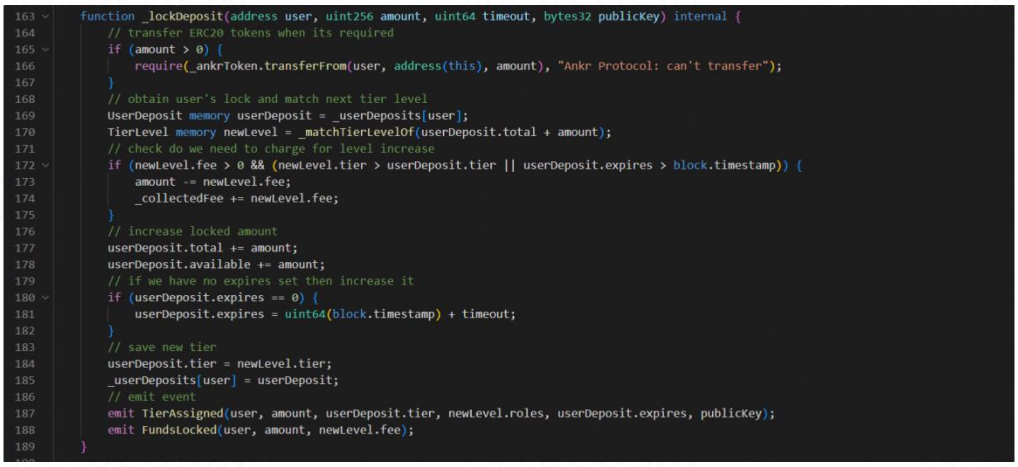


```
    function changeConsensus(address newValue) external onlyFromGovernance {
        require(newValue != address(0x00), "AnkrProtocol: zero address");
        address oldValue = _consensus;
        _consensus = newValue;
        emit ConsensusChanged(oldValue, newValue);
    }

    function changeGovernance(address newValue) external onlyFromGovernance {
        require(newValue != address(0x00), "AnkrProtocol: zero address");
        address oldValue = _governance;
        _governance = newValue;
        emit GovernanceChanged(oldValue, newValue);
    }

    function changeEnterpriseAdmin(address newValue) external onlyFromGovernance {
        require(newValue != address(0x00), "AnkrProtocol: zero address");
        address oldValue = newValue;
        _enterpriseAdmin = newValue;
        emit EnterpriseAdminChanged(oldValue, newValue);
    }
}
```

Figure 38 The source code of untouched functions(Fixed)

[AnkrProtocol-6] stake lock issue

Severity Level	Info
Type	Business Security
Lines	AnkrProtocol.sol #L163-189
Description	When the user stake, the timeout lock-up time is controlled by the user, and in the *_lockDeposit* function, the lock-up time is only updated when it is judged that userDeposit.expires == 0. Then the user can control the lock-up time and only lock the first stakes.



```
163     function _lockDeposit(address user, uint256 amount, uint64 timeout, bytes32 publicKey) internal {
164         // transfer ERC20 tokens when its required
165         if (amount > 0) {
166             require(_ankrToken.transferFrom(user, address(this), amount), "Ankr Protocol: can't transfer");
167         }
168         // obtain user's lock and match next tier level
169         UserDeposit memory userDeposit = _userDeposits[user];
170         TierLevel memory newLevel = _matchTierLevelOf(userDeposit.total + amount);
171         // check do we need to charge for level increase
172         if (newLevel.fee > 0 && (newLevel.tier > userDeposit.tier || userDeposit.expires > block.timestamp)) {
173             amount -= newLevel.fee;
174             _collectedFee += newLevel.fee;
175         }
176         // increase locked amount
177         userDeposit.total += amount;
178         userDeposit.available += amount;
179         // if we have no expires set then increase it
180         if (userDeposit.expires == 0) {
181             userDeposit.expires = uint64(block.timestamp) + timeout;
182         }
183         // save new tier
184         userDeposit.tier = newLevel.tier;
185         _userDeposits[user] = userDeposit;
186         // emit event
187         emit TierAssigned(user, amount, userDeposit.tier, newLevel.roles, userDeposit.expires, publicKey);
188         emit FundsLocked(user, amount, newLevel.fee);
189     }
```

Figure 39 The source code of *_lockDeposit* function

Recommendations	It is suggested that the lock-up period is fixed, and the lock-up start time is the user's stake time each time.
Status	Acknowledged.

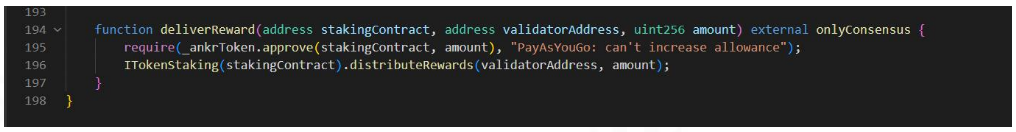
[PayAsYouGo-1] Reward source not specified

Severity Level Medium

Type Business Security

Lines PayAsYouGo.sol #L193-198

Description Consensus can call the *deliverReward* function to issue ankr token rewards to the current epoch of the stakingContract contract. This does not specify the source of the amount issued. If the amount is too large, the user's stake principal will be issued as a reward, which will cause losses to the user. If the source of the amount is the collected _collectedFee, then should judge whether the amount is less than _collectedFee and subtract the value of collectedFee in each call.

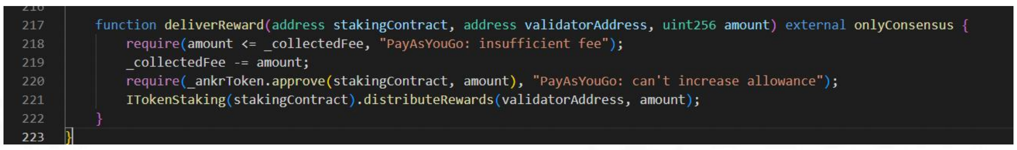


```
193
194     function deliverReward(address stakingContract, address validatorAddress, uint256 amount) external onlyConsensus {
195         require(_ankrToken.approve(stakingContract, amount), "PayAsYouGo: can't increase allowance");
196         ITokenStaking(stakingContract).distributeRewards(validatorAddress, amount);
197     }
198 }
```

Figure 40 The source code of *deliverReward* function

Recommendations It is recommended to distribute rewards from collectedFee and deduct the corresponding amount.

Status Fixed.



```
    function deliverReward(address stakingContract, address validatorAddress, uint256 amount) external onlyConsensus {
        require(amount <= _collectedFee, "PayAsYouGo: insufficient fee");
        _collectedFee -= amount;
        require(_ankrToken.approve(stakingContract, amount), "PayAsYouGo: can't increase allowance");
        ITokenStaking(stakingContract).distributeRewards(validatorAddress, amount);
    }
}
```

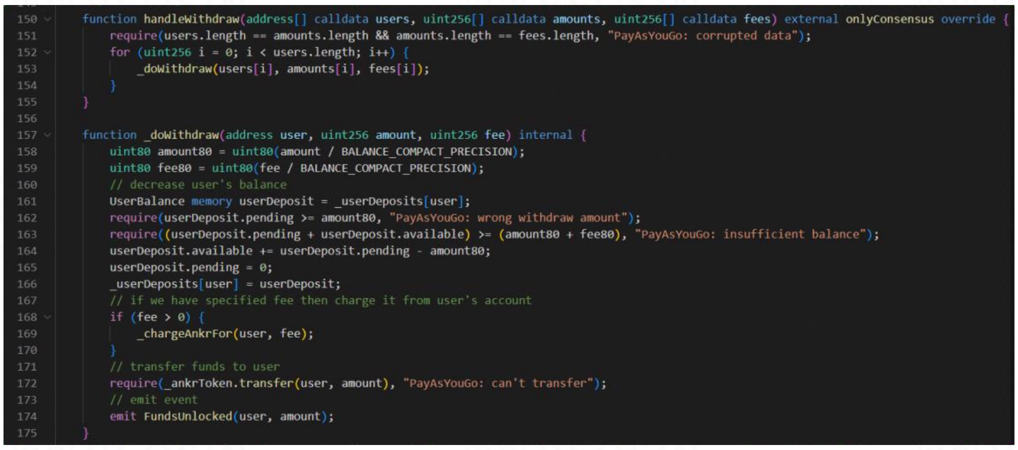
Figure 41 The source code of *deliverReward* function(Fixed)

[PayAsYouGo-2] User withdrawal issue

Severity Level	Low
Type	Business Security
Lines	PayAsYouGo.sol #L150-175

Description

The user can only call the *withdraw* function to increase the pending amount to be withdrawn, and the final transfer operation is implemented by Consensus calling the *handleWithdraw* function. The user's withdrawal amount and fee are not controlled by the user.



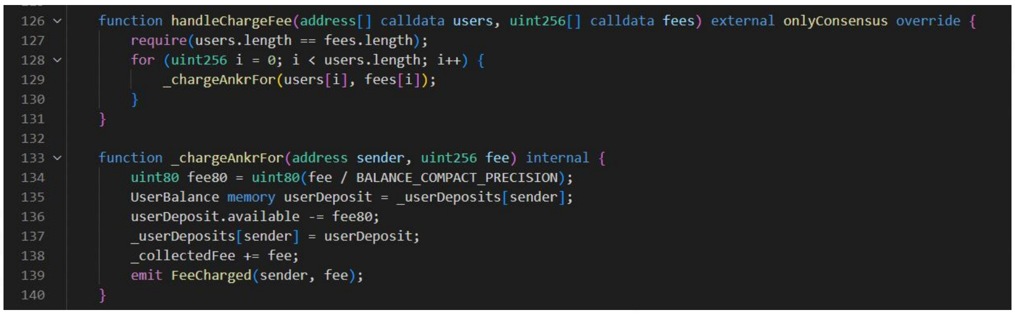
```
150    function handleWithdraw(address[] calldata users, uint256[] calldata amounts, uint256[] calldata fees) external onlyConsensus override {
151        require(users.length == amounts.length && amounts.length == fees.length, "PayAsYouGo: corrupted data");
152        for (uint256 i = 0; i < users.length; i++) {
153            _doWithdraw(users[i], amounts[i], fees[i]);
154        }
155    }
156
157    function _doWithdraw(address user, uint256 amount, uint256 fee) internal {
158        uint80 amount80 = uint80(amount / BALANCE_COMPACT_PRECISION);
159        uint80 fee80 = uint80(fee / BALANCE_COMPACT_PRECISION);
160        // decrease user's balance
161        UserBalance memory userDeposit = _userDeposits[user];
162        require(userDeposit.pending >= amount80, "PayAsYouGo: wrong withdraw amount");
163        require((userDeposit.pending + userDeposit.available) >= (amount80 + fee80), "PayAsYouGo: insufficient balance");
164        userDeposit.available += userDeposit.pending - amount80;
165        userDeposit.pending = 0;
166        _userDeposits[user] = userDeposit;
167        // if we have specified fee then charge it from user's account
168        if (fee > 0) {
169            _chargeAnkrFor(user, fee);
170        }
171        // transfer funds to user
172        require(_ankrToken.transfer(user, amount), "PayAsYouGo: can't transfer");
173        // emit event
174        emit FundsUnlocked(user, amount);
175    }
```

Figure 42 The source code of *handleWithdraw* and *_doWithdraw* functions

Recommendations	It is recommended to limit the value of fee within a reasonable range.
Status	Acknowledged.

[PayAsYouGo-3] Handling fee issue

Severity Level	**Low**
Type	Business Security
Lines	PayAsYouGo.sol #L126-140
Description	When the contract charges the fee through the *_chargeAnkrFor* function, the _collectedFee in the contract only increases but does not decrease. Then, the fee will not be processed in the contract.



```
126     function handleChargeFee(address[] calldata users, uint256[] calldata fees) external onlyConsensus override {
127         require(users.length == fees.length);
128         for (uint256 i = 0; i < users.length; i++) {
129             _chargeAnkrFor(users[i], fees[i]);
130         }
131     }
132 
133     function _chargeAnkrFor(address sender, uint256 fee) internal {
134         uint80 fee80 = uint80(fee / BALANCE_COMPACT_PRECISION);
135         UserBalance memory userDeposit = _userDeposits[sender];
136         userDeposit.available -= fee80;
137         _userDeposits[sender] = userDeposit;
138         _collectedFee += fee;
139         emit FeeCharged(sender, fee);
140     }
```

Figure 43 The source code of *handleChargeFee* and *_chargeAnkrFor* functions

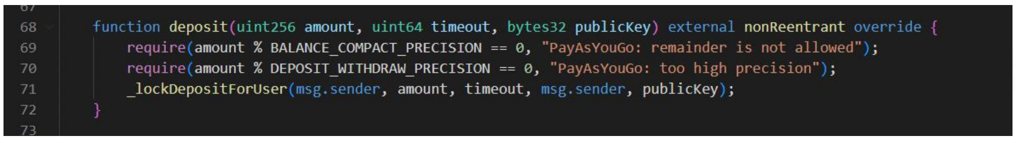
Recommendations	It is recommended to increase the extraction method of _collectedFee.
Status	Acknowledged. According to the description of the project party, fee here means not withdrawal fee, its fee for services. the project party doesn't charge fee immediately the project party charge it on weekly basis or on withdrawal. Its also intended.

[PayAsYouGo-4] Variable does not implement the relevant function

Severity Level	Info
Type	Business Security
Lines	PayAsYouGo.sol #L68-73, L93-103, L169-182

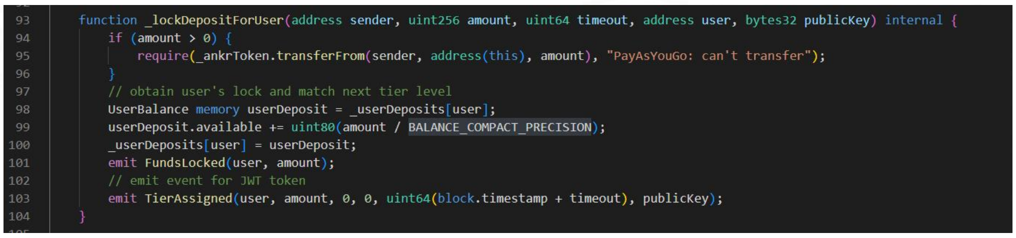
Description

The timeout and publicKey variables are passed in when the user stake s, but the contract is only used to trigger events and has no actual impact. When the user extracts, it just adds _requestNonce as a record and does not use it.



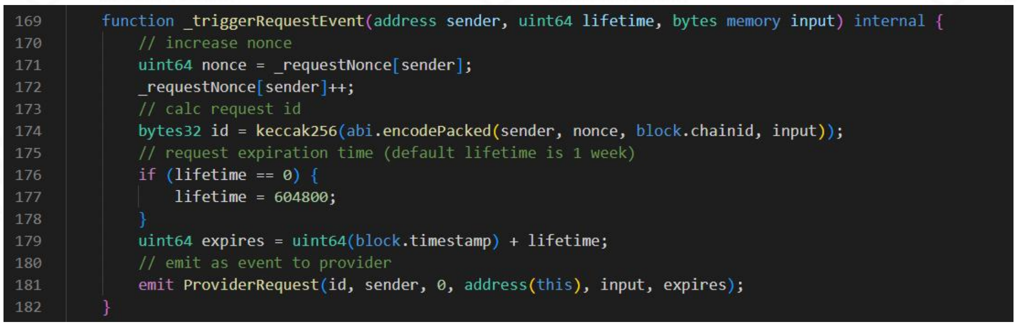
```
68    function deposit(uint256 amount, uint64 timeout, bytes32 publicKey) external nonReentrant override {
69        require(amount % BALANCE_COMPACT_PRECISION == 0, "PayAsYouGo: remainder is not allowed");
70        require(amount % DEPOSIT_WITHDRAW_PRECISION == 0, "PayAsYouGo: too high precision");
71        _lockDepositForUser(msg.sender, amount, timeout, msg.sender, publicKey);
72    }
73
```

Figure 44 The source code of *deposit* function



```
93    function _lockDepositForUser(address sender, uint256 amount, uint64 timeout, address user, bytes32 publicKey) internal {
94        if (amount > 0) {
95            require(_ankrToken.transferFrom(sender, address(this), amount), "PayAsYouGo: can't transfer");
96        }
97        // obtain user's lock and match next tier level
98        UserBalance memory userDeposit = _userDeposits[user];
99        userDeposit.available += uint80(amount / BALANCE_COMPACT_PRECISION);
100       _userDeposits[user] = userDeposit;
101       emit FundsLocked(user, amount);
102       // emit event for JWT token
103       emit TierAssigned(user, amount, 0, 0, uint64(block.timestamp + timeout), publicKey);
104   }
```

Figure 45 The source code of *_lockDepositForUser* function



```
169   function _triggerRequestEvent(address sender, uint64 lifetime, bytes memory input) internal {
170       // increase nonce
171       uint64 nonce = _requestNonce[sender];
172       _requestNonce[sender]++;
173       // calc request id
174       bytes32 id = keccak256(abi.encodePacked(sender, nonce, block.chainid, input));
175       // request expiration time (default lifetime is 1 week)
176       if (lifetime == 0) {
177           lifetime = 604800;
178       }
179       uint64 expires = uint64(block.timestamp) + lifetime;
180       // emit as event to provider
181       emit ProviderRequest(id, sender, 0, address(this), input, expires);
182   }
```

Figure 46 The source code of *_triggerRequestEvent* function

Recommendations	It is recommended to add related implementation.
Status	Acknowledged. According to the description of the project party, nonce is used to calculate request id. This event is used only to verify consensus of pending withdrawal to ask to process it.

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact / Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in Blockchain.

3.4 About BEOSIN

BEOSIN is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions.BEOSIN has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, BEOSIN has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com