BNB Side Chain Audit Report

Audit Period: 2022/09/12 - 2022/09/21

Overall Risk: Medium

Project links:

Github	https://github.com/node-real/semita-bas-template-bsc/pull/8 https://github.com/node-real/semita-bas-genesis-config/pull/11/

Audit Scope:

In the following, we show the SHA256 hash value of the compressed file used in this audit:

- SHA256 (semita-bas-genesis-config-netmarble_v1.0.zip) = a330b621805cc412a3d9a503e7cca10bd93601fe9eedb8fa07c07cb5c3d49bbf
- SHA256 (semita-bas-template-bsc-netmarble_v1.0.zip) = a7f6e03f7cba1ce9baa9b7ab391008d47500abe73022f27eedb1125ddf919e36

We found out the Audit Scope as:

- **Semita-bas-genesis-config-netmarble_v1.0**

The files in this folder include Genesis smart contracts and scripts for building the genesis config. By default, BNB Sidechain provides an EVM execution environment with a predefined set of system smart contracts for platform operation. We mainly audited the smart contract part, here it's some BNB Sidechain-defined smart contracts:

No.	Path	FileName	File Hash(SHA-256)	Verdict	Details
1	contracts	ChainConfig.sol	c3be0557947f8e8af0840d62a9f1491158d919ac3fbc821429ac5a71786a42eb	Informational	[I01] [I02] [I03] [Q01]
2		DeployerProxy.sol	ce4331ec3d14841d5075a753e8518a29fde7635fd40fb71966bdd9ec4d63ef75	Passed	
3		Governance.sol	776505f816a9c4cf74075b1ea088db39caa61b5c50cd12b	Informational	[I01] [Q01]

			7afabfab8f9722440		
4		InjectorContextHolder.sol	59cbc8adae75619c477c6249142c5771a928e1301b1fe9103c716c15653c9928	**Passed**	
5		Reserve.sol	bb487f38737d07a04e87264b5899c94b651a93cc702b0d79f2acbd932aab52a2	**Passed**	
6		Reward.sol	1d3f541196183f6ba3a73ec49afd70d0fa981465e1a2f1a23208938883e4cbab	**Informational**	**[I01]** **[Q05]**
7		RuntimeProxy.sol	fd40ac5d90fcb9ef807e2ce72688604be5d9c9a435ab666484c9e1c592666342	**Passed**	
8		RuntimeUpgrade.sol	407c9b4c24573cfe284a57194cede8e0b1b164a9f292ad606a99159586f0d862	**Informational**	**[I01]**
9		SlashingIndicator.sol	4572e0dfb21c03bd7153f3ee787fef46e15edec38f680cb7f24bdb1b1210dd39	**Passed**	
10		Staking.sol	b9f35efd61b4502075a6b3dd21729fc1c4c22eec21d0eeb31632a4054d68f59f	**Medium**	**[M01]** **[M02]** **[I01]** **[I02]** **[I03]** **[I04]** **[I05]** **[I06]** **[I07]** **[I08]** **[I09]** **[Q01]** **[Q02]** **[Q03]** **[Q04]** **[Q06]**
11		StakingPool.sol	7cd12d26b5f2fa7d913dfd7ff84c371b8b401f08e2782be78f	**Passed**	

			ddc67eb5027a0f		
12		SystemReward.sol	5bd0c2bac7b2402171b073ed937d78b6dcb6cef25ca41ec58c368367a557e8de	Informational	[I01]
13		TimeLock.sol	3b49fd1d9880b4b9e3a9154f7530fc1a1cd400af29684595f594ea078bc868a4	Informational	[I01]
14	contracts/libs	MultiCall.sol	05409ea9d759f56f3521eb0d80245358d64d240d3e9a19eb7c0c0de238aa3597	Passed	

- **Semita-bas-template-bsc-netmarble_v1.0**

No.	Path	FileName	File Hash(SHA-256)	Verdict	Details
1	contracts/checkpointoracle/contract	oracle.sol	7bacc9643706223fc231c9c52a9ddaac05918b2f375a59f2878b5abda952889d	Passed	

We also audited some of the non-contract parts of the BSC Application Sidechain. Since the BSC Application Sidechain is a fork from the BNB Smart chain, this audit mainly focuses on the code modification part of this sidechain. The file involved are:

No.	Path	FileName	File Hash(SHA-256)	Verdict	Details
1	common/systemcontract/	const.go	1d5904e1a07eb9527a99c7ce23011babf9457eafd008c7a595fc2923c915152c	Informational	[I01]
2	consensus/parlia/	parlia.go	78a3ea99c2bfd278dd95e915b685a92d929b50668a8822e4b3e7dcba510d8432	Informational	[I01]
3	core/vm/	contracts.go	71ab48d3048066f5c5aa299ea22741bafc67694b705d7e97d09285a47a5a0fa7	Informational	[I01] [I02]
4	core/vm/	evm.go	a11cf85b07a958cc0eb4eb6a5f5a25560310f0f9a0dd02f3954a722a9d27ca4f	Informational	[I01] [Q01]

5	core/state/	statedb.go	6e11fc9fa4f1f6812a621e6231ecb1787a815ee4cb299096587ee0d19b45812a	Informational	[I01]
6	core/	state_processor.go	57537a74ce0c441322282c066211058fb7dcd5fec8bf1352eecfca51fac6d7ea	Informational	[I01]
7	core/	tx_pool.go	a515eaab4a08eb42aaa6ef8c8e2f062b2bf3adbfa825fbfee087b40a360f6911	Low	[L01] [I01]
8	eth/	backend.go	e52785358d0236a28cd94a4ec70e48eeed1d7712fa7b0a144d6553d93aa1e34a	Informational	[I01]
9	internal/ethapi/	api.go	94ccf7c5ca0d54451554b0114b68f7380b59232b452b56ae59969b6453d294e7	Informational	[I01]
10	miner/	worker.go	81eff6d48d09a8bc66bcf985a263f406be68006d19d53ded1b236772957ef11	Informational	[I01]
11	p2p/dnsdisc/	sync.go	307e979ab5a08fe393124c680ba9901382b6d3b42761a6b8ac7402e3ca32b109	Informational	[I01]
12	params/	config.go	f595a24bc3cf2595c4c8dc815ee06055e8db9854566029bf2fd8da4ffe11db50	Informational	[I01]
13	params/	protocol_params.go	9be1e517e5e7d61aebdc58c1d18370d724829c14a87eb62691ecda246fb776a0	Informational	[I01]

Note:
1. The analysis of the security is purely based on the smart contracts mentioned in the Audit Scope.
2. Due to the time limit, the audit team did not do much in-depth research on the business logic of the project. It is more about discovering issues in the smart contracts itselves.

Semita-bas-genesis-config-netmarble_v1.0

1. ChainConfig.sol

Informational Severity

[I01] Missing Validation Check

In the *_setFreeGasAddressAdmin()* function, it is called by the Governance address to change the *freeGasAddressAdmin*.

```
185     function _setFreeGasAddressAdmin(address _freeGasAddressAdmin) internal {
186         require(_freeGasAddressAdmin != freeGasAddressAdmin, "Same admin!");
187         address temp = freeGasAddressAdmin;
188         freeGasAddressAdmin = _freeGasAddressAdmin;
189         emit FreeGasAddressAdminChanged(temp, freeGasAddressAdmin);
190     }
```

Even though there is a check that the new admin should not be the same as the previous admin, there is no check that the new admin should not be address “0x0”.

Suggestion: Add
`require(_freeGasAddressAdmin != address(0));`

[I02] Inconsistency for override keywords

Most of the functions implement the override keyword as they are overriding from the interface contract IChainConfig.sol. However, some functions that override do not implement the override keyword.
Besides, depending on which compiler version is used, override is no longer necessary from 0.8.8 and above.

Example with override keyword: *setUndelegatePeriod()*

```
154     function setUndelegatePeriod(uint32 newValue) external override onlyFromGovernance {
155         uint32 prevValue = _consensusParams.undelegatePeriod;
156         _consensusParams.undelegatePeriod = newValue;
157         emit UndelegatePeriodChanged(prevValue, newValue);
158     }
```

Example without override keyword: *getMinValidatorStakeAmount()*

```
160     function getMinValidatorStakeAmount() external view returns (uint256) {
161         return _consensusParams.minValidatorStakeAmount;
162     }
```

Suggestion:
It is better to ensure consistency across all functions.

[I03] Gas Optimization

In the function "_removeFreeGasAddress" assigning the value of "_freeGasAddressList.length" to a temporary variable and subsequently replacing the usage of "_freeGasAddressList.length" with the temporary variable will help save gas.

```
ftrace | funcSig
219     function _removeFreeGasAddress(address freeGasAddress⬆) internal {
220         uint256 position = _freeGasAddressMap[freeGasAddress⬆];
221         // remove freeGasAddress
222         if (position > 0) {
223             uint256 indexOf = position - 1;
224             if (_freeGasAddressList.length > 1 && indexOf != _freeGasAddressList.length - 1) {
225                 address lastAddress =  _freeGasAddressList[_freeGasAddressList.length - 1];
226                 _freeGasAddressList[indexOf] = lastAddress;
227                 _freeGasAddressMap[lastAddress] = indexOf + 1;
228             }
229             _freeGasAddressList.pop();
230             delete _freeGasAddressMap[freeGasAddress⬆];
231             emit FreeGasAddressRemoved(freeGasAddress⬆);
232         }
233     }
```

Suggestion: Modify lines 222-225 to

```
if (position > 0) {
      uint256 indexOf = position - 1;
      GAListlength = _freeGasAddressList.length
      if (GAListlength > 1 && indexOf != GAListlength - 1) {
            address lastAddress =  _freeGasAddressList[GAListlength -
      1];
```

3. Governance.sol

Informational Severity

[I01] Code Clarity

In this function, *getVotingPower()* is supposed to return the voting power of an owner's validator address.

```
49      function getVotingPower(address validator) external view returns (uint256) {
50          return _validatorOwnerVotingPowerAt(validator, block.number);
51      }
```

Thus, *_validatorOwnerVotingPowerAt()* is called which will get the validator address to subsequently get its voting power.

```
110     function _validatorOwnerVotingPowerAt(address validatorOwner, uint256 blockNumber) ir
111         address validator = _STAKING_CONTRACT.getValidatorByOwner(validatorOwner);
112         return _validatorVotingPowerAt(validator, blockNumber);
```

However, the calling function uses input parameter *validator* which is misleading as the input should be *validatorOwner.*

Suggestion: Modify to
`function getVotingPower(address validatorOwner)`

6. Reward.sol

Informational Severity

[I01] Missing Non-Zero Check

Non-zero checks are used throughout the codebase (such as in SystemReward._claimSystemFee()) to prevent the unintentional sending of funds to the 0x0 address. As shown below, In Reward.sol the function *burnAndRelease()* shows a clear distinction between funds designated to be burned, and funds allocated to the foundation.

```
132     function burnAndRelease() external {
133         uint256 balance = address(this).balance;
134         uint256 burned = balance * burnRatio / RATIO_SCALE;
135         uint256 released = burned * releaseRatio / RATIO_SCALE;
136
137         if (address(_RESERVE_CONTRACT).balance >= released) {
138             payable(deadAddress).transfer(burned);
139             uint256 unburned = balance - burned;
140             payable(foundationAddress).transfer(unburned);
141             _RESERVE_CONTRACT.release(foundationAddress, released);
142             emit BurnedAndReserveReleased(burned, foundationAddress, unburned + released);
143         } else {
144             payable(foundationAddress).transfer(balance);
145             emit BurnedAndReserveReleased(0, foundationAddress, balance);
146         }
147     }
```

However neither the function *burnAndRelease()*, nor *updateFoundationAddress()* have a non-zero check to ensure that funds allocated to the foundation are not mistakenly burned.

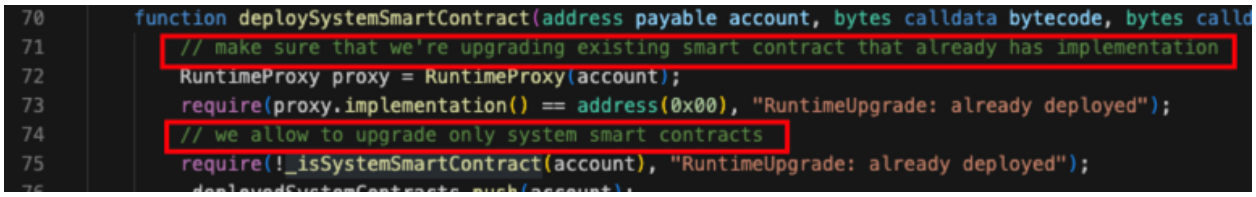
```
67      function updateFoundationAddress(address _foundationAddress⬆) public onlyThis {
68          address preValue = foundationAddress;
69          foundationAddress = _foundationAddress⬆;
70          emit UpdateFoundationAddress(preValue, _foundationAddress⬆);
71      }
```

8. RuntimeUpgrade.sol

Informational Severity

[I01] Wrong comments

In the *deploySystemSmartContract()* function (deploy a new logic contract, upgrade), the comments here are copied from the above function *upgradeSystemSmartContract()* (upgrade a proxy to a new logic contract). As such, the comments here are inconsistent with the code.



Suggestion: Change the comments to

```
// make sure that we're deploying a new smart contract that does not have implementation
// make sure the smart contract is not an existing system smart contract
```

10. Staking.sol

Medium Severity

[M01] Business Logic

In the *deposit()* payable function, the *_depositFee()* payable function is called, where the miner deposits a fee/mining reward to a particular validator. Hence, the total rewards will be according to the msg.value.

```
808     function deposit(address validatorAddress) external payable onlyFromCoinbase virtual override {
809         _depositFee(validatorAddress);
810     }
811
812     // DONE
813     function _depositFee(address validatorAddress) internal {
814         require(msg.value > 0);
815         _safeTransferWithGasLimit(payable(address(_REWARD_CONTRACT)), msg.value);
816         // make sure validator is active
817         Validator memory validator = _validatorsMap[validatorAddress];
818         require(validator.status != ValidatorStatus.NotFound, "not found");
819         uint64 epoch = currentEpoch();
820         // increase total pending rewards for validator for current epoch
821         ValidatorSnapshot storage currentSnapshot = _touchValidatorSnapshot(validator, epoch);
822         currentSnapshot.totalRewards += 0;
823         // emit event
824         emit ValidatorDeposited(validatorAddress, msg.value, epoch);
825     }
```

However, in line 822, the currentSnapshot for the validator's totalRewards += 0, which essentially has no effect.

```
821         ValidatorSnapshot storage currentSnapshot = _touchValidatorSnapshot(validator, epoch);
822         currentSnapshot.totalRewards += 0;
```

Suggestion: To amend the business logic to `currentSnapshot.totalRewards += msg.value;`

[M02] Business Logic - Strict Equality

In line 922, there is a strict equality check where a validator is put in jail if his *slashesCount* == *felonyThreshold*.

```
914     // slash and put in Jail
915     function slash(address validatorAddress) external onlyFromSlashingIndicator virtual override {
916         _slashValidator(validatorAddress);
917     }
```

```
910     function _slashValidator(address validatorAddress) internal {
911         // make sure validator exists
912         Validator memory validator = _validatorsMap[validatorAddress];
913         require(validator.status != ValidatorStatus.NotFound, "not found");
914         uint64 epoch = currentEpoch();
915
916         // increase slashes for current epoch
917         ValidatorSnapshot storage currentSnapshot = _touchValidatorSnapshot(validator, epoch);
918         uint32 slashesCount = currentSnapshot.slashesCount + 1;
919         currentSnapshot.slashesCount = slashesCount;
920
921         // if validator has a lot of misses then put it in jail for 1 week (if epoch is 1 day)
922         if (slashesCount == _CHAIN_CONFIG_CONTRACT.getFelonyThreshold()) {
923             validator.jailedBefore = currentEpoch() + _CHAIN_CONFIG_CONTRACT.getValidatorJailEpochLength()
```

However, consider this situation where the *felonyThreshold* is 2 and this particular validator has 1 slashCount. Now, if the felonyThreshold were to be updated to 1 in the ChainConfig.sol contract. If the _slashValidator function is called now, since the slashesCount is incremented first, then the check will now be bypassed.
This is because his slashesCount is now 2 and the felonyThreshold is now 1.

What if the *slashesCount* > *_CHAIN_CONFIG_CONTRACT.getFelonyThreshold()*? What will happen? It seems the validator.status will not be changed into the Jail state again. Will the Governance role call the removeValidator() method to remove that validator manually?

Suggestion:
It is recommended to review the business logic to prevent such a situation from happening.

Informational Severity

[I01] Gas Optimization

In the function *_removeValidatorfromActiveList()* assigning the value of *_activeValidatorssList.length* to a temporary variable and subsequently replacing the usage of *_activeValidatorssList.length* with the temporary variable will help save gas.
It is understood that the length of the array iterated across represents the number of validators on a particular side-chain. Therefore there may possibly be a high amount of iterations, given a highly decentralized side-chain. Due to BSC having implemented EIP-2929, following the suggestion, each iteration will save 100 gas.

```
604     function _removeValidatorFromActiveList(address validatorAddress⬆) internal {
605         // find index of validator in validator set
606         int256 indexOf = - 1;
607         for (uint256 i = 0; i < _activeValidatorsList.length; i++) {
608             if (_activeValidatorsList[i] != validatorAddress⬆) continue;
609             indexOf = int256(i);
610             break;
611         }
612         // remove validator from array (since we remove only active it might not exist in the list)
613         if (indexOf >= 0) {
614             if (_activeValidatorsList.length > 1 && uint256(indexOf) != _activeValidatorsList.length - 1) {
615                 _activeValidatorsList[uint256(indexOf)] = _activeValidatorsList[_activeValidatorsList.length - 1];
616             }
617             _activeValidatorsList.pop();
618         }
619     }
```

[I02] Code Clarity

In Staking.sol, there exists a mapping *_validatorsMap* to map a validator address to the structure Validator.

```
113     // mapping from validator address to validator
114     mapping(address => Validator) internal _validatorsMap;
```

```
87    struct Validator {
88        address validatorAddress;
89        address ownerAddress;
90        ValidatorStatus status;
91        uint64 changedAt;
92        uint64 jailedBefore;
93        uint64 claimedAt;
94    }
```

This mapping is used across several functions such as *activateValidator()* and *getValidatorFee()*, where the parameter used is *validatorAddress*.

```
622    function activateValidator(address validatorAddress) external onlyFromGovernance virtual override {
623        Validator memory validator = _validatorsMap[validatorAddress];
```

However, in functions like *removeValidator()*, *isValidatorActive()* and *isValidator()* the parameter used is *account:*

```
590    function removeValidator(address account) external onlyFromGovernance virtual override {
591        Validator memory validator = _validatorsMap[account];
```

Suggestion:
Use `_validatorsMap[validatorAddress]` for consistency.

[I03] Redundant Comments

In the function *_delegateTo()*, the *amount* parameter is checked to be more than or equal to the min staking amount(explained in line 319's comment). Line 322 is a redundant comment as it states the same thing and can be removed.

```
318    function _delegateTo(address fromDelegator, address toValidator, uint256 amount) inte
319        // check is minimum delegate amount
320        require(amount >= _CHAIN_CONFIG_CONTRACT.getMinStakingAmount() && amount != 0, "
321        require(amount % BALANCE_COMPACT_PRECISION == 0, "no remainder");
322        // make sure amount is greater than min staking amount
323        // make sure validator exists at least
```

[I04] Wrong Comments

In this function _undelegateFrom(), the goal is to decrease the delegation amount from the validator. Therefore, the comment at line 387 is supposed to be:
`// decrease total delegated amount…`

```
378     function _undelegateFrom(address toDelegator, address fromValidator, uint256 amount) internal {
379         // check minimum delegate amount
380         require(amount >= _CHAIN_CONFIG_CONTRACT.getMinStakingAmount() && amount != 0, "too low");
381         require(amount % BALANCE_COMPACT_PRECISION == 0, "no remainder");
382         // make sure validator exists at least
383         Validator memory validator = _validatorsMap[fromValidator];
384         uint64 beforeEpoch = nextEpoch();
385         // Lets upgrade next snapshot parameters:
386         // + find snapshot for the next epoch after current block
387         // + increase total delegated amount in the next epoch for this validator
388         // + re-save validator because last affected epoch might change
389         ValidatorSnapshot storage validatorSnapshot = _touchValidatorSnapshot(validator, beforeEpoch);
390         require(validatorSnapshot.totalDelegated >= uint112(amount / BALANCE_COMPACT_PRECISION), "insufficient balance");
391         validatorSnapshot.totalDelegated -= uint112(amount / BALANCE_COMPACT_PRECISION);
392         _validatorsMap[fromValidator] = validator;
```

The comments at lines 394-396 are also wrong. They should be:

```
// 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, otherwise there are no pending delegations and should create a record in delegateQueue
// create new record in undelegateQueue with the last epoch (undelegations are ordered by epoch)
```

```
394         // if last pending delegate has the same next epoch then its safe to just increase total
395         // staked amount because it can't affect current validator set, but otherwise we must create
396         // new record in delegation queue with the last epoch (delegations are ordered by epoch)
397         ValidatorDelegation storage delegation = _validatorDelegations[fromValidator][toDelegator];
398         require(delegation.delegateQueue.length > 0, "insufficient balance");
399
400         DelegationOpDelegate storage recentDelegateOp = delegation.delegateQueue[delegation.delegateQueue.length - 1];
401         require(recentDelegateOp.amount >= uint64(amount / BALANCE_COMPACT_PRECISION), "insufficient balance");
```

[I05] Inconsistency With Comment

In the *_depositFee()* function, there is a check in line 782 to check if the validator exists at least. However, line 780's comment checks if the validator is active. Therefore, the comment and code implementation is inconsistent.

```
777     function _depositFee(address validatorAddress) internal {
778         require(msg.value > 0);
779         _safeTransferWithGasLimit(payable(address(_REWARD_CONTRACT)), msg.value);
780         // make sure validator is active
781         Validator memory validator = _validatorsMap[validatorAddress];
782         require(validator.status != ValidatorStatus.NotFound, "not found");
```

```
75      enum ValidatorStatus {
76          NotFound,
77          Active,
78          Pending,
79          Jail
80      }
```

[I06] Unnecessary Condition

The internal function *_redelegateDelegatorRewards()* is only called once in this contract by the *redelegateDelegatorFee()* function. The last 2 parameters being passed through are *true* and *false*.

```
894     function redelegateDelegatorFee(address validator) external override {
895         // claim rewards in the redelegate mode (check function code for more info)
896         _redelegateDelegatorRewards(validator, msg.sender, currentEpoch(), true, false);
897     }
```

As such, the condition for withUndelegates in lines 442 and 443 are redundant and can be removed.

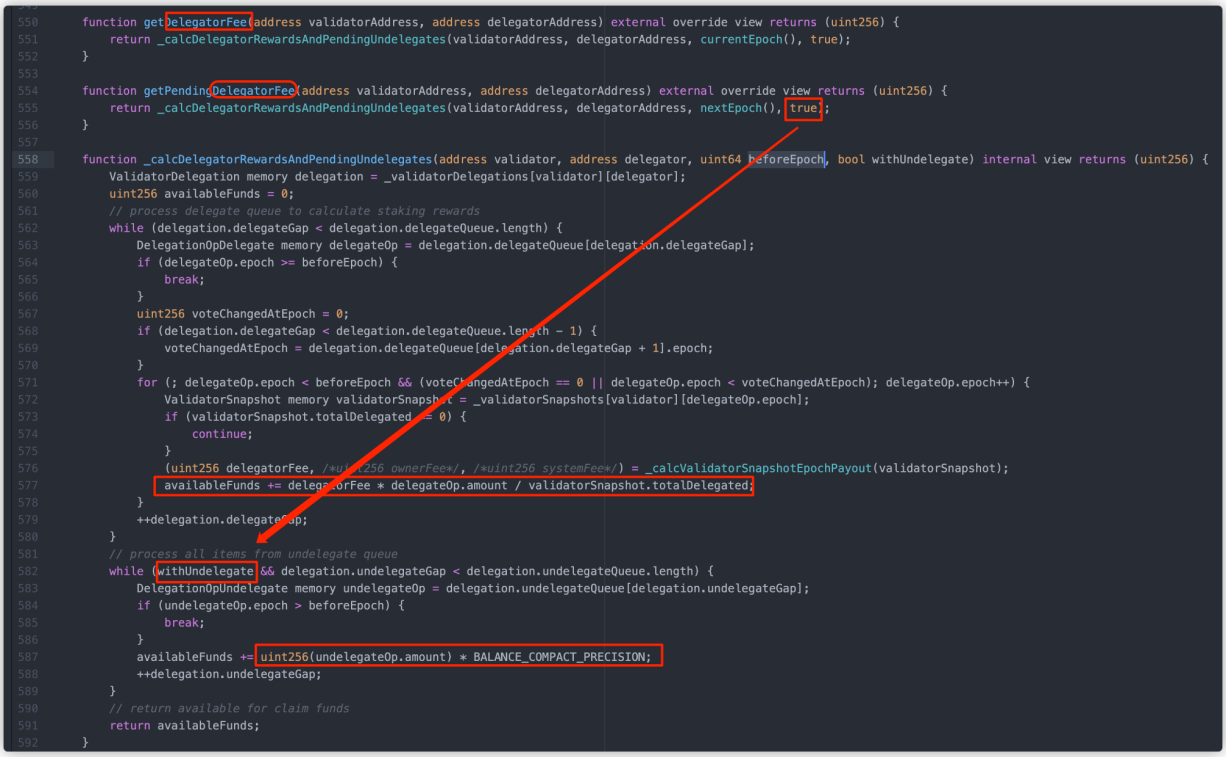
```
435     function _redelegateDelegatorRewards(address validator, address delegator, uint64 beforeEpochExclude, bool withRewards, bool withUndelegate
436         ValidatorDelegation storage delegation = _validatorDelegations[validator][delegator];
437         // claim rewards and undelegates
438         uint256 availableFunds = 0;
439         if (withRewards) {
440             availableFunds += _processDelegateQueue(validator, delegation, beforeEpochExclude);
441         }
442         if (withUndelegates) {
443             availableFunds += _processUndelegateQueue(delegation, beforeEpochExclude);
```

[I07] Wrong Error Message

The error message within the red box of the *_undelegateFrom()* internal function is not reasonable.

```
function _undelegateFrom(address toDelegator, address fromValidator, uint256 amount) internal {
    // check minimum delegate amount
    require(amount >= _CHAIN_CONFIG_CONTRACT.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");
    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}));
    }
    // create new undelegate queue operation with soft lock
    delegation.undelegateQueue.push(DelegationOpUndelegate({amount : uint112(amount / BALANCE_COMPACT_PRECISION), epoch : beforeEpoch + _CHAIN_CONFIG_CONTRACT.getUndelegatePeriod()}));
    // emit event with the next epoch number
    emit Undelegated(fromValidator, toDelegator, amount, beforeEpoch);
}
```

[I08] Inconsistency Between Function Name And Function Logic



```
550     function getDelegatorFee(address validatorAddress, address delegatorAddress) external override view returns (uint256) {
551         return _calcDelegatorRewardsAndPendingUndelegates(validatorAddress, delegatorAddress, currentEpoch(), true);
552     }
553
554     function getPendingDelegatorFee(address validatorAddress, address delegatorAddress) external override view returns (uint256) {
555         return _calcDelegatorRewardsAndPendingUndelegates(validatorAddress, delegatorAddress, nextEpoch(), true);
556     }
557
558     function _calcDelegatorRewardsAndPendingUndelegates(address validator, address delegator, uint64 beforeEpoch, bool withUndelegate) internal view returns (uint256) {
559         ValidatorDelegation memory delegation = _validatorDelegations[validator][delegator];
560         uint256 availableFunds = 0;
561         // process delegate queue to calculate staking rewards
562         while (delegation.delegateGap < delegation.delegateQueue.length) {
563             DelegationOpDelegate memory delegateOp = delegation.delegateQueue[delegation.delegateGap];
564             if (delegateOp.epoch >= beforeEpoch) {
565                 break;
566             }
567             uint256 voteChangedAtEpoch = 0;
568             if (delegation.delegateGap < delegation.delegateQueue.length - 1) {
569                 voteChangedAtEpoch = delegation.delegateQueue[delegation.delegateGap + 1].epoch;
570             }
571             for (; delegateOp.epoch < beforeEpoch && (voteChangedAtEpoch == 0 || delegateOp.epoch < voteChangedAtEpoch); delegateOp.epoch++) {
572                 ValidatorSnapshot memory validatorSnapshot = _validatorSnapshots[validator][delegateOp.epoch];
573                 if (validatorSnapshot.totalDelegated == 0) {
574                     continue;
575                 }
576                 (uint256 delegatorFee, /*uint256 ownerFee*/, /*uint256 systemFee*/) = _calcValidatorSnapshotEpochPayout(validatorSnapshot);
577                 availableFunds += delegatorFee * delegateOp.amount / validatorSnapshot.totalDelegated;
578             }
579             ++delegation.delegateGap;
580         }
581         // process all items from undelegate queue
582         while (withUndelegate && delegation.undelegateGap < delegation.undelegateQueue.length) {
583             DelegationOpUndelegate memory undelegateOp = delegation.undelegateQueue[delegation.undelegateGap];
584             if (undelegateOp.epoch > beforeEpoch) {
585                 break;
586             }
587             availableFunds += uint256(undelegateOp.amount) * BALANCE_COMPACT_PRECISION;
588             ++delegation.undelegateGap;
589         }
590         // return available for claim funds
591         return availableFunds;
592     }
```

From the function names, the return value of the *getDelegatorFee()* and *getPendingDelegatorFee()* functions only includes *DelegatorFee*, but in fact, it also includes the amount of undelegated funds.

[I09] Ambiguous Function Name

For the *getValidators()* function, it will only return validators in the Active status. The validators in Pending or Jail status will not be included.

```
function getValidators() public view override returns (address[] memory) {
    uint256 n = _activeValidatorsList.length;
    address[] memory orderedValidators = new address[](n);
    for (uint256 i = 0; i < n; i++) {
        orderedValidators[i] = _activeValidatorsList[i];
    }
    // we need to select k top validators out of n
    uint256 k = _CHAIN_CONFIG_CONTRACT.getActiveValidatorsLength();
    if (k > n) {
        k = n;
    }
    for (uint256 i = 0; i < k; i++) {
        uint256 nextValidator = i;
        Validator memory currentMax = _validatorsMap[orderedValidators[nextValidator]];
        ValidatorSnapshot memory maxSnapshot = _validatorSnapshots[currentMax.validatorAddress][currentMax.changedAt];
        for (uint256 j = i + 1; j < n; j++) {
            Validator memory current = _validatorsMap[orderedValidators[j]];
            ValidatorSnapshot memory currentSnapshot = _validatorSnapshots[current.validatorAddress][current.changedAt];
            if (maxSnapshot.totalDelegated < currentSnapshot.totalDelegated) {
                nextValidator = j;
                currentMax = current;
                maxSnapshot = currentSnapshot;
            }
        }
        address backup = orderedValidators[i];
        orderedValidators[i] = orderedValidators[nextValidator];
        orderedValidators[nextValidator] = backup;
    }
    // this is to cut array to first k elements without copying
    assembly {
        mstore(orderedValidators, k)
    }
    return orderedValidators;
}
```

Suggestion:

It is recommended to change the function name of *getValidators()* function to *getActiveValidators()* to avoid ambiguity.

12. SystemReward.sol

Informational Severity

[I01] Gas Optimization

In the function *_claimSystemFee()*, one can save gas by assigning the value of *_distributionShares.length* to a temporary variable and subsequently replacing the usage of *_distributionShares.length* with the temporary variable.

```
132         for (uint256 i = 0; i < _distributionShares.length; i++) {
133             DistributionShare memory ds = _distributionShares[i];
134             uint256 accountFee = amountToPay * ds.share / SHARE_MAX_VALUE;
135             payable(ds.account).transfer(accountFee);
136             emit FeeClaimed(ds.account, accountFee);
137             totalPaid += accountFee;
138         }
139         // return some dust back to the acc
140         _systemFee = amountToPay - totalPaid;
```

13. SystemReward.sol

Informational Severity

[I01] Missing Non-Zero Check

In the *__TimeLock_init_unchained()* function, the admin is set to a particular address. This internal function is called in the *Reward.initialize()* function. However, there is no check for address(0) here. This can render the contract useless.

```
61      function __TimeLock_init_unchained(address admin_, uint256 delay_) internal onlyInitializing {
62          require(delay_ >= MINIMUM_DELAY, "Timelock::constructor: Delay must exceed minimum delay.");
63          require(delay_ <= MAXIMUM_DELAY, "Timelock::constructor: Delay must not exceed maximum delay.");
64
65          admin = admin_;
66          delay = delay_;
67      }
```

Suggestion:
It is recommended to include check:
`require(admin != address(0));`

Questions:

[Q01] Across several contracts in the codebase, we noticed that many functions incorporate the ‘virtual’ keyword. Is there a reason for incorporating that?

Across several contracts like ChainConfig.sol, Staking.sol, and Governance.sol, many functions have the ‘virtual’ keyword added.

Examples:

```
215     function removeFreeGasAddress(address freeGasAddress) external onlyFromFreeGasAddressAdmin virtual override
216         _removeFreeGasAddress(freeGasAddress);
217     }
```

```
808     function deposit(address validatorAddress) external payable onlyFromCoinbase virtual override {
809         _depositFee(validatorAddress);
810     }
```

```
55      function proposeWithCustomVotingPeriod(
56          address[] memory targets,
57          uint256[] memory values,
58          bytes[] memory calldatas,
59          string memory description,
60          uint256 customVotingPeriod
61      ) public virtual onlyValidatorOwner(msg.sender) returns (
```

However, at the moment, It is unclear where they will be overridden. As such, the ‘virtual’ keyword can be removed if unused.

[Q02] In Staking.sol, the validator address is an EOA while the validator owner should also be an EOA. Does the *owner* here mean that he controls the validator address?

```
115     // mapping from validator owner to validator address
116     mapping(address => address) internal _validatorOwners;
```

```
ftrace | funcSig
556 function addValidator(address account⬆) external onlyFromGovernance virtual override {
557     _addValidator(account⬆, account⬆, ValidatorStatus.Active, 0, 0, nextEpoch());
558 }
559
ftrace | funcSig
560 function _addValidator(address v                                                    int16 commissionRate⬆, uint25
561     // validator commission rate
562     require(commissionRate⬆ >= COMMISSION_RATE_MIN_VALUE && commissionRate⬆ <= COMMISSION_RATE_MAX_VALUE, "bad commission");
563     // init validator default params
564     Validator memory validator = _validatorsMap[validatorAddress⬆];
565     require(_validatorsMap[validatorAddress⬆].status == ValidatorStatus.NotFound, "already exist");
566     validator.validatorAddress = validatorAddress⬆;
567     validator.ownerAddress = validatorOwner⬆;
568     validator.status = status⬆;
569     validator.changedAt = sinceEpoch⬆;
570     _validatorsMap[validatorAddress⬆] = validator;
571     // save validator owner
572     require(_validatorOwners[validatorOwner⬆] == address(0x00), "owner in use");
573     _validatorOwners[validatorOwner⬆] = validatorAddress⬆;
574     // add new validator to array
575     if (status⬆ == ValidatorStatus.Active) {
576         _activeValidatorsList.push(validatorAddress⬆);
577     }
578     // push initial validator snapshot at zero epoch with default params
579     _validatorSnapshots[validatorAddress⬆][sinceEpoch⬆] = ValidatorSnapshot(0, uint112(initialStake⬆ / BALANCE_COMPACT_PRECISION),
580     // delegate initial stake to validator owner
581     ValidatorDelegation storage delegation = _validatorDelegations[validatorAddress⬆][validatorOwner⬆];
582     require(delegation.delegateQueue.length == 0);
583     delegation.delegateQueue.push(DelegationOpDelegate(uint112(initialStake⬆ / BALANCE_COMPACT_PRECISION), sinceEpoch⬆));
584     emit Delegated(validatorAddress⬆, validatorOwner⬆, initialStake⬆, sinceEpoch⬆);
585     // emit event
586     emit ValidatorAdded(validatorAddress⬆, validatorOwner⬆, uint8(status⬆), commissionRate⬆);
587 }
588
```

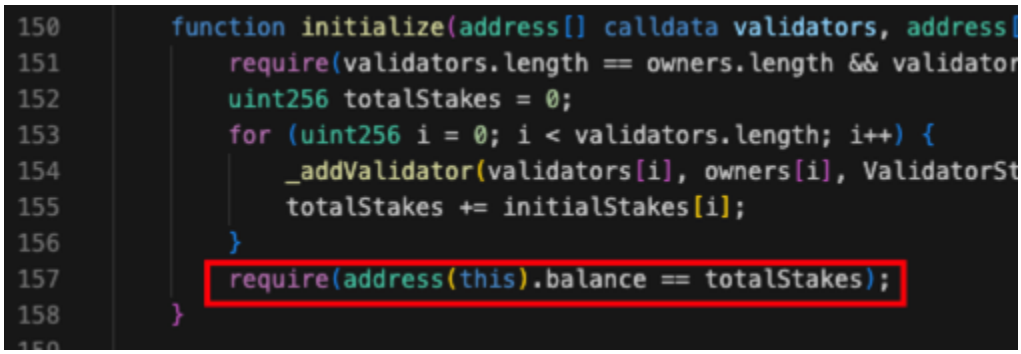
CONST (*uint16*) StateVar *Staking*.COMMISSION_RATE_MIN_VALUE = 0 (Declaration: #43)

```
uint16 internal constant COMMISSION_RATE_MIN_VALUE = 0;
```

[Q03] In Staking.sol, the initialize function has a check at the end for balance. Does this mean the validators need to send enough balance in first before initialization?

Also, curious if the address(this).balance is BNB or the native sidechain token. Our understanding is the native sidechain token.

```
153     function initialize(address[] calldata validators, address[] calldata owners, uint256[] calldata initialStakes, uint16 commissionRate) external initializer {
154         require(validators.length == owners.length && validators.length == initialStakes.length);
155         uint256 totalStakes = 0;
156         for (uint256 i = 0; i < validators.length; i++) {
157             _addValidator(validators[i], owners[i], ValidatorStatus.Active, commissionRate, initialStakes[i], 0);
158             totalStakes += initialStakes[i];
159         }
160         require(address(this).balance == totalStakes);
161     }
```



[Q04] Certik brought up an issue CON-01, which was only acknowledged.

Exceed Block Gas Limit

Category	Severity	Location	Status
Volatile Code	Critical	Staking.sol (v3): 412, 421, 460, 469; StakingPool.sol (v3): 88, 96	Acknowledged

Description

The identified loop logic in `_processDelegateQueue()` and `_calcDelegatorRewardsAndPendingUndelegates()` may exceed block gas limit when parameter `beforeEpochExclude`/`beforeEpoch` is far bigger than the epoch of last processed delegation. It can happen if functions `redelegateDelegatorFee()`/`claimDelegatorFee()`/`claimDelegatorFeeAtEpoch()` have not been called by a delegator for a long time. This is especially critical for `StakingPool` because it does not use `claimDelegatorFeeAtEpoch()` to process delegation little by little. Thus the modifier `advanceStakingRewards()` in `StakingPool` may always fail due to the use of `redelegateDelegatorFee()`/`calcAvailableForRedelegateAmount()`. Then the functions `stake()`/`unstake()`/`claim()` in `StakingPool` will always fail and users' funds are locked in the contract and lost.

To avoid this risk, maybe the project team should inform users to process delegates frequently.

[Q05] In Reward.sol, is foundationAddress the Staking.sol contract?

```
function burnAndRelease() external {
    uint256 balance = address(this).balance;
    uint256 burned = balance * burnRatio / RATIO_SCALE;
    uint256 released = burned * releaseRatio / RATIO_SCALE;

    if (address(_RESERVE_CONTRACT).balance >= released) {
        payable(deadAddress).transfer(burned);
        uint256 unburned = balance - burned;
        payable(foundationAddress).transfer(unburned);
        _RESERVE_CONTRACT.release(foundationAddress, released);
        emit BurnedAndReserveReleased(burned, foundationAddress, unburned +
    } else {
        payable(foundationAddress).transfer(balance);
        emit BurnedAndReserveReleased(0, foundationAddress, balance);
    }
```

[Q06] What's the meaning of re-delegate to the same validator?

```
522     function _redelegateDelegatorRewards(address validator, address delegator, uint64 beforeEpochExclude, bool withRewards, bool withUndelegates) internal {
523         ValidatorDelegation storage delegation = _validatorDelegations[validator][delegator];
524         // claim rewards and undelegates
525         uint256 availableFunds = 0;
526         if (withRewards) {
527             availableFunds += _processDelegateQueue(validator, delegation, beforeEpochExclude);
528         }
529         if (withUndelegates) {
530             availableFunds += _processUndelegateQueue(delegation, beforeEpochExclude);
531         }
532         (uint256 amountToStake, uint256 rewardsDust) = _calcAvailableForRedelegateAmount(availableFunds);
533         // if we have something to re-stake then delegate it to the validator
534         if (amountToStake > 0) {
535             _delegateTo(delegator, validator, amountToStake);
536         }
537         // if we have dust from staking then send it to user (we can't keep them in the contract)
538         if (rewardsDust > 0) {
539             _safeTransferWithGasLimit(payable(delegator), rewardsDust);
540         }
541         // emit event
542         emit Redelegated(validator, delegator, amountToStake, rewardsDust, beforeEpochExclude);
543     }
544
545     function redelegateDelegatorFee(address validator) external override {
546         // claim rewards in the redelegate mode (check function code for more info)
547         _redelegateDelegatorRewards(validator, msg.sender, currentEpoch(), true, false);
548     }
```

In some other POS projects, delegators re-delegate to a new validator sometimes to avoid slashing or get higher returns. But here delegators can only re-delegate to the same validator in the *_redelegateDelegatorRewards()* method of the Staking.sol, what is the intention of this design?

semita-bas-template-bsc-netmarble_v1.0

Common Issue

Informational Severity

[I01] Fork Low Version

This side chain is fork version 1.1.8 of the BNB Smart chain. The latest version of the BNB Smart chain is 1.1.13. Although there is no public vulnerability information, we can see from the changelog of the BNB Smart chain that a lot of bugs have been fixed, see https://github.com/bnb-chain/bsc/blob/master/CHANGELOG.md for details.

Therefore, it is recommended to fork the side chain code from the latest version of the BNB Smart chain code, which will reduce potential bugs.

3. contracts.go

Informational Severity

[I02] Redundant Code

In the core/vm/contracts.go, in the *RunPrecompiledContract()* function, there is such a piece of code:

```
156     // special case for EVM hooks (they consume 0 gas), and we should be 100% sure that we use copy of the input
157     if gasCost == 0 {
158         newInput := make([]byte, len(input))
159         copy(newInput, input)
160         input = newInput
161     }
```

In terms of function, this is a piece of redundant code. What is the meaning of this code?

7. tx_pool.go

Low Severity

[L01] Inconsistency Between Two Arrays

Within the *reset()* function of the core/tx_pool.go, we can see in the end it called the *removeGasFree(address) method*:

```
1273    gasFreeAddressMap, err := pool.gasFreeAddressMapFunc(pool.chain.CurrentBlock().Hash())
1274    if err != nil {
1275        log.Warn("Failed to get gasFreeAddressMap", "err", err)
1276    } else {
1277        pool.gasFreeAddressMap = gasFreeAddressMap
1278        for address, _ := range pool.locals.gasFreeAccounts {
1279            _, exist := pool.gasFreeAddressMap[address]
1280            if !exist {
1281                log.Debug("Remove gasFreeAccount", "account", address.Hex())
1282                pool.locals.removeGasFree(address)
1283            }
1284        }
1285    }
```

Analyze the function logic of removeGasFree(address), if calling removeGasFree(addr) function, the input-parameter addr will be deleted from both gasFreeAccounts and accounts. But throughout the whole code, accounts will not perform deletion operations, so it is believed that the removeGasFree() operation will cause the account to be out of sync in the two arrays.

```go
// add inserts a new address into the set to track.
func (as *accountSet) add(addr common.Address) {
    as.accounts[addr] = struct{}{}
    as.cache = nil
}
func (as *accountSet) addGasFree(addr common.Address) {
    as.accounts[addr] = struct{}{}
    as.gasFreeAccounts[addr] = struct{}{}
    as.cache = nil
}

func (as *accountSet) removeGasFree(addr common.Address) {
    delete(as.accounts, addr)
    delete(as.gasFreeAccounts, addr)
    as.cache = nil
}

// addTx adds the sender of tx into the set.
func (as *accountSet) addTx(tx *types.Transaction) {
    if addr, err := types.Sender(as.signer, tx); err == nil {
        as.add(addr)
    }
}
```

Questions:

[Q01] For the core/vm/evm.go, In the Line #562, what is the purpose of using OPCODE STOP to create a contract?

```go
// CreateWithAddress creates a new contract using code as deployment code.
func (evm *EVM) CreateWithAddress(caller common.Address, code []byte, gas uint64, value *big.Int, contractAddr common.Address) (ret []byte, leftOverGas uint64, err error) {
    ret, _, leftOverGas, err = evm.create(AccountRef(caller), &codeAndHash{code: code}, gas, value, contractAddr, STOP)
    return
}
```