hexens × PERSISTENCE

APR.24

SECURITY REVIEW REPORT FOR PERSISTENCE

CONTENTS

- Unable to initiate unbonding due to an underflow error with the weightage variable
- Unable to execute StakePool.epochUpdate() because of the mismatch between shares and stakes of a validator
- Selfdestruct is deprecated after the Kepler hardfork
- Unnecessary if statement in the DelegationManager.delegateDepositedBNB() function
- Lack of tests
- Deposits can temporarily stuck in DelegationManager contract
- Function _isActiveValidator() does not check the state of StakeHub contract
- Validator status is not checked in the initiateDelegation() function
- Redundant conditions are present in the check function of the ValidatorSet contract
- Redundant imports

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered an update to stkBNB, Persistence's Liquid Staking solution. The update is a migration of all modules from the Beacon Chain to the Binance Smart Chain.

Our security assessment was a full review of new smart contracts and changes made to existing smart contracts, spanning a total of 2 weeks.

During our audit, we identified 1 critical severity vulnerability in the redelegation mechanism that could cause shares to get lost. We also identified 7 high severity vulnerabilities.

Moreover, we have also identified several minor severity vulnerabilities and code optimisations.

Finally, all of our reported issues were fixed by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

SCOPE

The analyzed resources are located on:

https://github.com/persistenceOne/stkBNB-contracts/tree/amrith/bc-fusion-v2/contracts

The issues described in this report were fixed in the following commit:

https://github.com/persistenceOne/stkBNB-contracts/pull/46/commits/28c5398c1f391617b64d6606723bb1f9554e4009

AUDITING DETAILS

STARTED
01.04.2024

DELIVERED
12.04.2024

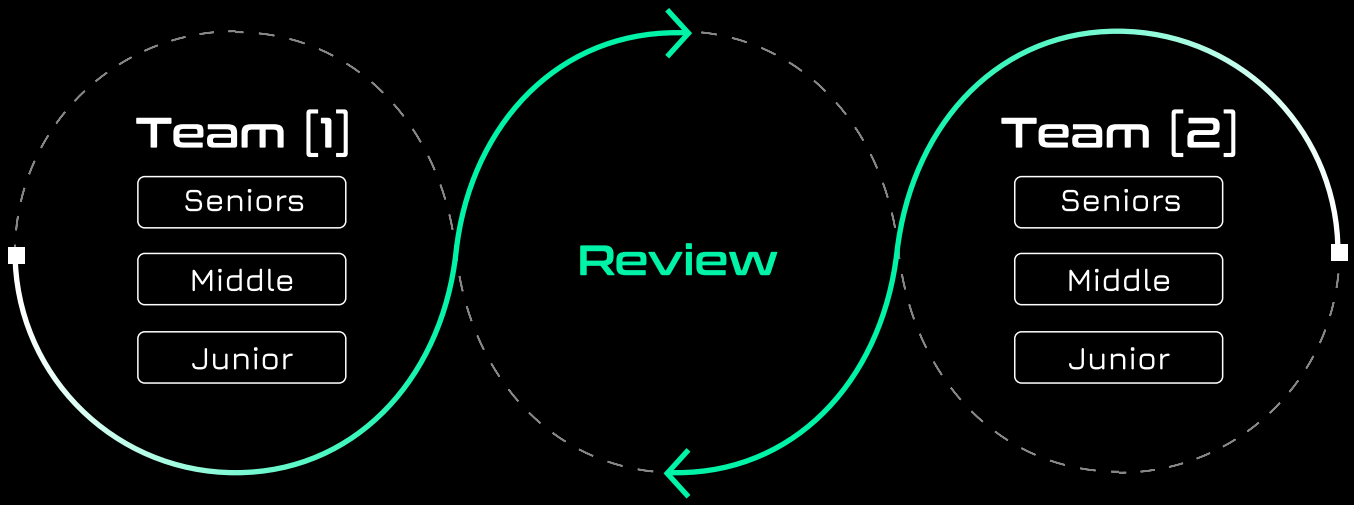
Review Led by

TRUNG DINH

Security Researcher | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.



SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

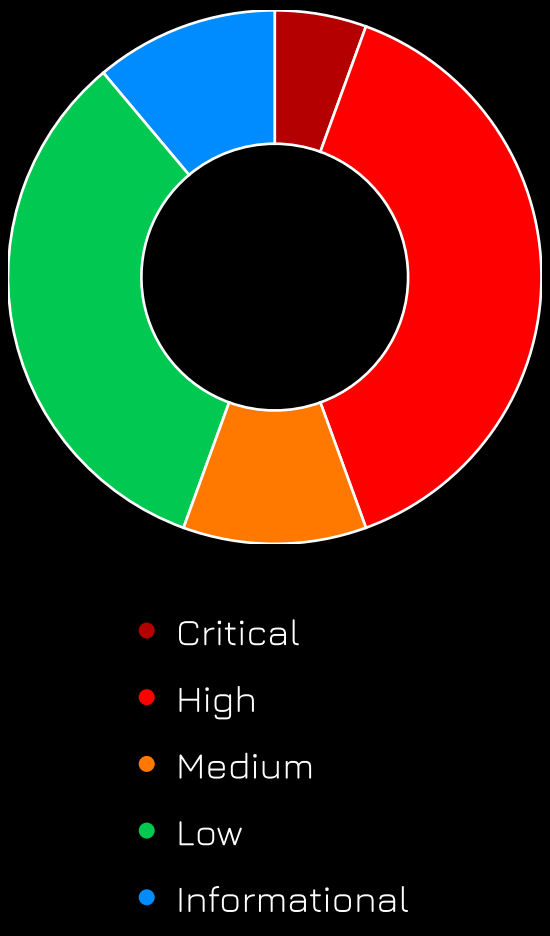
ISSUE SYMBOLIC CODES

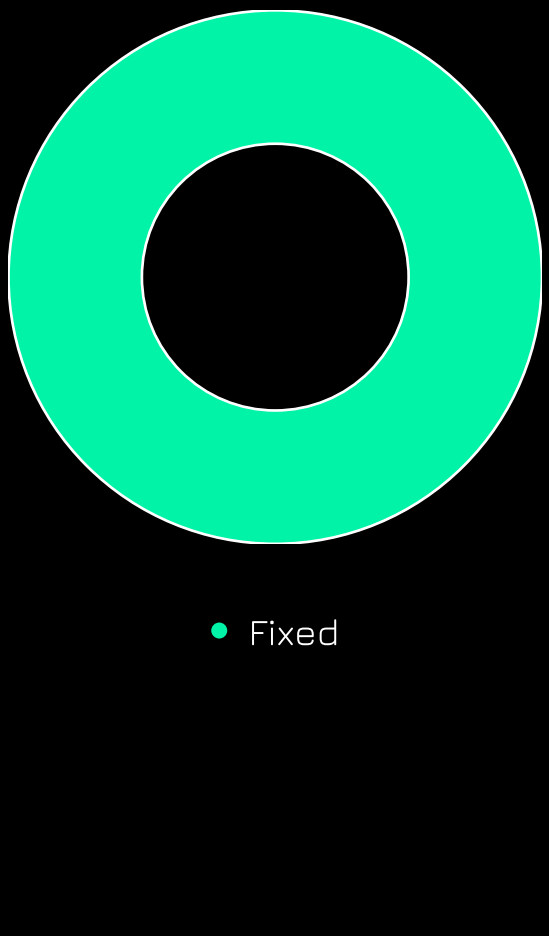
Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	1
High	7
Medium	2
Low	6
Informational	2

Total: 18





WEAKNESSES

This section contains the list of discovered weaknesses.

PRST2-2

INCORRECTLY USING THE SHARES OF THE DESTINATION VALIDATOR AS THE INPUT FOR THE _STAKE_HUB.REDELEGATE() FUNCTION

SEVERITY: Critical

PATH:

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L775-L781

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L784-L813

REMEDIATION:

Consider using the shares of the source validator instead.

STATUS: Fixed

DESCRIPTION:

Within the **StakePool.initiateRedelegation()** function, if the **allotment** falls within the range of 1000 to 9000, the **partialShares** variable determines the amount of shares to be added to the **dstOperator_**. This quantity is then forwarded to the **DelegationManager.redelegateBnbShares()** function as the **shares** input, which is subsequently utilized for the external call to **_STAKE_HUB.redelegate()**.

However, in the built-in Stake Hub contract on the BSC chain, the **redelegate()** function requires the shares input to represent the number of **shares** to be removed from the **srcValidator** rather than the shares to be added to the **dstValidator**.

```
function redelegate(
    address srcValidator,
    address dstValidator,
    uint256 shares,
    bool delegateVotePower
)
    external
    whenNotPaused
    notInBlackList
    validatorExist(srcValidator)
    validatorExist(dstValidator)
    enableReceivingFund
{
    if (shares == 0) revert ZeroShares();
    if (srcValidator == dstValidator) revert SameValidator();


    address delegator = msg.sender;
    Validator memory srcValInfo = _validators[srcValidator];
    Validator memory dstValInfo = _validators[dstValidator];
            if   (dstValInfo.jailed   &&   delegator   !=   dstValidator)   revert
OnlySelfDelegation();


    uint256 bnbAmount = IStakeCredit(srcValInfo.creditContract).unbond(delegator,
shares);

    ...
}
```

This discrepancy leads to an incorrect amount of shares being burned from the **srcValidator**, causing a mismatch between the internal accounting **_validatorStore.validators[].shares** in the **StakePool** contract and the actual shares held by the **DelegationManager** contract.

It's important to note that this issue also arises when **allotment** equals 10000, as the **dstDelegation.shares** value is passed to the **DelegationManager.redelegateBnbShares()** function instead of **srcValidator.delegation.shares**.

In the worst-case scenario, when **allotment** equals 10000 and **srcValidator.delegation.shares** exceeds **dstDelegation.shares**, not all shares from the **srcValidator** are burned. However, the accounting for **srcValidator.delegation.shares** is zero after the function call, resulting in some shares remaining locked in the contract.

```
if (allotment_ == 10_000) {
    // 100 % of the stakes are redelegated
    dstDelegation = ValidatorSet.DelegationInfo({
        stakes: srcValidator.delegation.stakes,
        shares: _getValidatorShares(dstOperator_,
srcValidator.delegation.stakes),
        weightage: srcValidator.delegation.weightage
    });


    dstValidator._redelegate(
        srcValidator,
        dstDelegation,
        srcValidator.delegation,
        getTotalValidators()
    );


    shares = dstDelegation.shares;

IDelegationManager(payable(_addressStore.getDelegationManager())).redelegateBnbSh
ares(
        srcOperator_,
        dstOperator_,
        shares,
        false
    );
} else if (allotment_ >= 1_000 && allotment_ <= 9_000) {
    // Partial amount of stakes are redelegated (between 10 % to 90 %)
    uint256 partialShares = _getValidatorShares(
        dstOperator_,
        _rateFactor(srcValidator.delegation.stakes, allotment_)
    );
    dstDelegation = ValidatorSet.DelegationInfo({
        stakes: _rateFactor(srcValidator.delegation.stakes, allotment_),
        shares: partialShares,
        weightage: _rateFactor(srcValidator.delegation.weightage, allotment_)
    });
```

```
    srcDelegation = ValidatorSet.DelegationInfo({
        stakes: _rateFactor(srcValidator.delegation.stakes, allotment_),
        shares: _rateFactor(srcValidator.delegation.shares, allotment_),
        weightage: _rateFactor(srcValidator.delegation.weightage, allotment_)
    });


    dstValidator._redelegate(
        srcValidator,
        dstDelegation,
        srcDelegation,
        getTotalValidators()
    );


    shares = partialShares;

IDelegationManager(payable(_addressStore.getDelegationManager())).redelegateBnbSh
ares(
        srcOperator_,
        dstOperator_,
        shares,
        false
    );
```

PRST2-3

THE STAKEPOOL.INITIATEREDELEGATION() FUNCTION DOES NOT FACTOR IN THE REDELEGATION FEE

SEVERITY: High

PATH:

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L762-L766

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L788-L792

REMEDIATION:

Consider taking fee into account when calculating the shares and stakes of the dstValidator in function StakePool.initiateRedelegation().

STATUS: Fixed

DESCRIPTION:

In the built-in Stake Hub contract, the **StakeHub.redelegate()** function imposes a fee for this operation.

```
function redelegate(
    address srcValidator,
    address dstValidator,
    uint256 shares,
    bool delegateVotePower
)
    external
    whenNotPaused
    notInBlackList
    validatorExist(srcValidator)
    validatorExist(dstValidator)
    enableReceivingFund
{
    ...

            uint256 feeCharge = bnbAmount * redelegateFeeRate /
REDELEGATE_FEE_RATE_BASE;
     (bool success,) = dstValInfo.creditContract.call{ value: feeCharge }
("");
    if (!success) revert TransferFailed();

    bnbAmount -= feeCharge;

    ...
}
```

This implies that not all the BNB amounts removed from the **srcValidator** will be utilized for delegating to the **dstValidator**. However, in the **StakePool.initiateRedelegation()** function, the increased stakes and shares of the **dstValidator** are summed up with the corresponding percentage of **srcValidator.delegation.stakes** without considering the fee. This oversight in fee consideration results in incorrect accounting of the stakes and shares for the validators involved in the redelegation process.

```
        dstDelegation = ValidatorSet.DelegationInfo({
            stakes: srcValidator.delegation.stakes,
                                shares:   _getValidatorShares(dstOperator_,
srcValidator.delegation.stakes),
            weightage: srcValidator.delegation.weightage
        });
```

```
        dstDelegation = ValidatorSet.DelegationInfo({
            stakes: _rateFactor(srcValidator.delegation.stakes, allotment_),
            shares: partialShares,
                   weightage:  _rateFactor(srcValidator.delegation.weightage,
allotment_)
        });
```

PRST2-1

INDEX OUT OF BOUNDS ERROR IN FUNCTION STAKEPOOL._GETDAILYREWARD()

SEVERITY: High

PATH:

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L1112-L1115

REMEDIATION:

Consider modifying line 1112 to

for (uint256 i = 0; i < getTotalValidators(); ++i) {

STATUS: Fixed

DESCRIPTION:

The function **StakePool._getDailyRewards()** is used to compute the daily rewards earned. It accomplishes this by iterating through all the validators listed in the **_validatorStore.operatorList[]** array. However, an issue arises when the loop iterates from **0** to **_validatorStore.operatorList.length** instead of **_validatorStore.operatorList.length - 1**, causing an "Index out of bounds" error when attempting to access the element **_validatorStore.operatorsList[_validatorStore.operatorList.length]**.

The function **_getDailyReward()** is called within the **StakePool.epochUpdate()** function, responsible for updating user rewards. Due to this error, the function fails to execute, preventing users from receiving staking rewards.

```
    for (uint256 i = 0; i <= getTotalValidators(); ++i) {
        ValidatorSet.Info storage validator = _validatorStore.validators[
            _validatorStore.operatorsList[i]
        ];
```

PRST2-4

UTILIZATION OF OUTDATED VALIDATOR STAKES DURING REDELEGATION

SEVERITY: High

PATH:

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L762-L766

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L788-L798

REMEDIATION:

Consider updating the delegation.stakes for the srcValidator and dstValidator at the beginning of the function StakePool.initiateRedelegation().

STATUS: Fixed

DESCRIPTION:

In the **StakePool.initiateRedelegation()** function, when **allotment_ == 10000**, all stakes from the **srcValidator** are intended to be redelegated to the **dstValidator**. Following this redelegation, the accounting **stakes** of the **dstValidator** should increase by the amount of BNB held by the **srcValidator**, thereby leaving no BNB held by the **srcValidator** after the process.

However, this assertion does not hold true if the **srcValidator** receives distributed rewards in the **StakeCredit** contract but the **_getDailyRewards()** function has not been triggered in the **StakePool** contract.

In such a scenario, within the **initiateRedelegation()** function, the actual amount of BNB held by the **srcValidator** exceeds the value of **srcValidator.delegation.stakes** due to reward distribution. Consequently, not all shares from the **srcDelegator** are burned after the delegation process, although the values of **srcValidator.delegation.stakes** and **srcValidator.delegation.shares** both become zero after the call. This results in a portion of **shares** from the **srcValidator** becoming locked in the contract.

In summary, the described call flow would result in the locking of shares belonging to the **srcValidator**:

1. StakeCredit.distributeReward()
2. StakePool.initiateRedelegation()
3. StakePool.epochUpdate()

Please note that the issue also pertains to cases where **allotment_** does not equal 10000, potentially resulting in a mismatch between the internal accounting of stakes and shares for the validator and the actual amount of BNB held by that validator.

```
dstDelegation = ValidatorSet.DelegationInfo({
    stakes: srcValidator.delegation.stakes,
                              shares:       _getValidatorShares(dstOperator_,
srcValidator.delegation.stakes),
    weightage: srcValidator.delegation.weightage
});
```

```
dstDelegation = ValidatorSet.DelegationInfo({
    stakes: _rateFactor(srcValidator.delegation.stakes, allotment_),
    shares: partialShares,
    weightage: _rateFactor(srcValidator.delegation.weightage, allotment_)
});


srcDelegation = ValidatorSet.DelegationInfo({
    stakes: _rateFactor(srcValidator.delegation.stakes, allotment_),
    shares: _rateFactor(srcValidator.delegation.shares, allotment_),
    weightage: _rateFactor(srcValidator.delegation.weightage, allotment_)
});
```

PRST-2-7

BROKEN VALIDATOR CREATION LOGIC

SEVERITY: High

PATH:
8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L631

REMEDIATION:

Replace the line 631 in the file contracts/StakePool.sol with the following code to check that operator_ and stCred_ variables do not equal to zero address:

if (operator_ == _ZERO_ADDR || stCred_ == _ZERO_ADDR) revert ZeroAddress();

STATUS: Fixed

DESCRIPTION:

The current condition in the **_createValidator** function of the **StakePool** contract on line **631** checks that the operator address and the validator credit contract address are not equal to the zero address at the same time, otherwise a revert occurs with the **ZeroAddress** error:

```
if (operator_ != _ZERO_ADDR && stCred_ != _ZERO_ADDR) revert ZeroAddress();
```

This logic is incorrect because following situation occurs:

1. It is not possible to add an existing validator created previously through **StakeHub**, since the **getValidatorCreditContract** function of the **StakeHub** builtin contract will return a non-zero address for the existing credit contract and **_createValidator** will revert with **ZeroAddress** error

2. If a validator for the specified operator does not exist in **StakeHub**, the credit contract address (**stCred_** variable) will remain zero and will bypass the first **ZeroAddress** check. However, it is still not possible to add the validator to the **_validatorStore.validators** structure, since later the function **_checkCreate** in the ValidatorSet library checks that the credit contract address isn't zero

As a result, in current conditions it is not possible to add a validator.

```
function _createValidator(address operator_) private returns (bool) {
                                                address      stCred_      =
IStakeHub(_STAKE_HUB).getValidatorCreditContract(operator_);
            if (operator_ != _ZERO_ADDR && stCred_ != _ZERO_ADDR) revert
ZeroAddress(); // @audit wrong condition


                                                                    if        (!
_validatorStore.validators[operator_]._isNotValidator(getTotalValidators()))
{
            revert ValidatorAlreadyExists();
        } else {
            // Creates new Validator
            _validatorStore.validators[operator_]._create(
                operator_,
                stCred_,
                getTotalValidators() + 1
            );
            _validatorStore.operatorsList.push(operator_);
        }

        return true;
    }
```

```
  function _getDailyRewards() internal returns (uint256) {
          ...
          for (uint256 i = 0; i <= getTotalValidators(); ++i) {
                                      ValidatorSet.Info   storage   validator   =
  _validatorStore.validators[
                  _validatorStore.operatorsList[i]
              ];
              // Rewards accured by a validator on a specific day
                                                 uint256    validatorReward    =
  IStakeCredit(validator.stCred).getPooledBNB(
                  _addressStore.getDelegationManager()
              ) - validator.delegation.stakes;
              ...
          }
          ...
      }
```

contracts/StakePool.sol:L1145-1156

```
function _getValidatorShares(
        address _operator,
        uint256 _bnbAmount
    ) internal view returns (uint256) {
                                                                              if
(_validatorStore.validators[_operator]._isNotValidator(getTotalValidators()))
            revert ValidatorDoesNotExists();
        // Validator Credit Contract Address
        address stCred = _validatorStore.validators[_operator].stCred;
        uint256 shares = IStakeCredit(stCred).getSharesByPooledBNB(_bnbAmount);

        return shares;
    }
```

PRST2-13

INCORRECT USAGE OF SHARES INSTEAD OF STAKES OCCURS WHEN CALLING THE VALIDATORSET._REDELEGATE FUNCTION

SEVERITY: High

PATH:
524d33878fb37bb584516199bb6d997c5536c7c0/contracts/StakePool.sol:L829

REMEDIATION:

Modifying line 829 of contract StakePool.sol to

dstValidator._redelegate(srcValidator, dstStakes, srcStakes);

STATUS: Fixed

DESCRIPTION:

The function **ValidatorSet._redelegate()** is utilized to update the internal accounting of the amount of BNBs held by each validator after the redelegation process. The function accepts two parameters, **dstStakes** and **srcStakes**, which correspondingly represent the increase in the amount of BNBs for the destination validator and the decrease in the amount of BNBs for the source stakes.

Both of these variables should be in the form of BNB amounts rather than validator shares. However, in the function **StakePool.initiateRedelegation()**, when the **allotment_** falls within the range of **[1000, 9000]**, the function **ValidatorSet._redelegate()** is invoked in line 829 with the **srcShares**, which denotes the amount of shares removed from the source validator.

This erroneous input passage causes the internal accounting of the stakes for the source validator to be larger than expected (as shares are always less than stakes). Consequently, when the function **StakePool.epochUpdate()** is triggered, the calculated **validatorReward** for the source validator in line 1146 is lower than anticipated. This results in a smaller value of **bnbReward** being returned from the internal function **_getDailyReward()**. Consequently, the **totalWei** update is less than anticipated, leading to subsequent deposit or withdraw functions minting or burning an incorrect amount of shares.

```
dstValidator._redelegate(srcValidator, dstStakes, srcShares);
```

PRST2-14

INCORRECT VALUE OF EXCHANGERATE.TOTALWEI DUE TO REDELEGATION FEES

SEVERITY: High

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/StakePool.sol:L780-L835

REMEDIATION:

Consider updating the exchangeRate.totalWei following the redelegation fee.

STATUS: Fixed

DESCRIPTION:

Within the **StakePool.initiateRedelegation()** function, there is no update for the **exchangeRate.totalWei**, which implies that the function operates under the assumption that there will be no changes to the amount of BNBs held by the **DelegationManager** in the relegation process. However, this assumption is incorrect because during the redelegation process, the **StakeHub** built-in contract also imposes a fee. As a result, the actual amount of BNBs held by the **DelegationManager** decreases after the process, causing the accounting for **exchangeRate.totalWei** to be greater than expected following the redelegation.

```
function initiateRedelegation(
    address srcOperator_,
    address dstOperator_,
    uint256 allotment_
) external override whenNotPaused onlyRole(BOT_ROLE) {
    if (!getValidator(srcOperator_)._isActiveValidator()) {
        revert ValidatorDoesNotExist();
    }


    if (!getValidator(dstOperator_)._isActiveValidator()) {
        _createValidator(dstOperator_);
    }


    ValidatorSet.Info storage srcValidator =
_validatorStore.validators[srcOperator_];
    ValidatorSet.Info storage dstValidator =
_validatorStore.validators[dstOperator_];


    uint256 sharesAlloted;
    if (allotment_ == 10_000) {
        // 100 % of the stakes are redelegated
        sharesAlloted = _getValidatorShares(srcOperator_,
srcValidator.delegation.stakes);

IDelegationManager(payable(_addressStore.getDelegationManager())).redelegateBnbSh
ares(
            srcOperator_,
            dstOperator_,
            sharesAlloted,
            false
        );


        uint256 dstStakes = IStakeCredit(dstValidator.stCred).getPooledBNB(
            _addressStore.getDelegationManager()
        );
```

```
        dstValidator._redelegate(srcValidator, dstStakes,
srcValidator.delegation.stakes);
        srcValidator.status = ValidatorSet.Status.Inactive;
    } else if (allotment_ >= 1_000 && allotment_ <= 9_000) {
        // Partial amount of stakes can be redelegated (between 10 % to 90 %)
        uint256 srcStakes = _rateFactor(srcValidator.delegation.stakes,
allotment_);
        uint256 srcShares = _getValidatorShares(srcOperator_, srcStakes);



IDelegationManager(payable(_addressStore.getDelegationManager())).redelegateBnbSh
ares(
            srcOperator_,
            dstOperator_,
            srcShares,
            false
        );


        uint256 dstStakes = IStakeCredit(dstValidator.stCred).getPooledBNB(
            _addressStore.getDelegationManager()
        );


        dstValidator._redelegate(srcValidator, dstStakes, srcShares);
    } else {
        revert InvalidAllotment(allotment_);
    }


    emit Redelegation_Success(srcOperator_, dstOperator_, block.timestamp);
}
```

PRST2-15

THE TOTALWEI IS INFLATED DUE TO THE PRESENCE OF DUPLICATED VALIDATOR ADDRESSES

SEVERITY: High

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/StakePool.sol:L622-L651

REMEDIATION:

Consider allowing the creation of only non-existing validators.

STATUS: Fixed

DESCRIPTION:

The **StakePool.createValidator()** function permits the Bot role to create an existing validator provided that the validator's status is not ACTIVE. In such cases, the function still adds the validator to the storage array **_validatorStore.operatorsList**, resulting in two validators with the same address being included in the **operatorList**. This duplication poses a risk for the **StakePool.epochUpdate()** function. The **epochUpdate()** function iterates through the entire **operatorList** array and calculates the **validatorReward** for each element. With two elements having the same address, the BNB reward will be counted twice, inflating the total reward amount and causing the **exchangeRate.totalWei** to be larger than expected.

```
    } else {
        // Creates new Validator
        uint256 prevStake =
validatorCredit.getPooledBNB(_addressStore.getDelegationManager());
        uint256 prevUnstake = validatorCredit.lockedBNBs(
            _addressStore.getDelegationManager(),
            0
        );


        ValidatorSet.Info memory newValidator = prevStake > 0
            ? ValidatorSet.Info({
                operator: operator_,
                stCred: stCred_,
                lastStakedAt: block.timestamp,
                delegation: ValidatorSet.DelegationInfo({
                    stakes: prevStake,
                    unstakes: prevUnstake
                }),
                status: ValidatorSet.Status.Active
            })
            : ValidatorSet.Info({
                operator: operator_,
                stCred: stCred_,
                lastStakedAt: 0,
                delegation: ValidatorSet.DelegationInfo({ stakes: 0, unstakes: 0
}),
                status: ValidatorSet.Status.Active
            });


        _validatorStore.validators[operator_]._create(newValidator);
        _validatorStore.operatorsList.push(operator_);
    }
```

PRST2-6

UNABLE TO INITIATE UNBONDING DUE TO AN UNDERFLOW ERROR WITH THE WEIGHTAGE VARIABLE

SEVERITY: Medium

PATH:

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L849-L856

REMEDIATION:

Consider removing the weightage variable in struct DelegationInfo if not necessary.

STATUS: Fixed

DESCRIPTION:

To withdraw BNBs from the **StakePool** contract, a user typically follows three steps:

- STEP-1: Transfer **stkBNB** to the StakePool contract to activate the internal function **_withdraw()**, which reduces the value of **exchange.totalWei**.
- STEP-2: Wait for the BOT_ROLE to complete the unbonding process (**unbondingInitiated()** -> **unbondingFinished**), which transfers the BNBs from the validators to the **DelegationManager**.
- STEP-3: Invoke the **claim()** function to receive the requested BNB.

The issue arises when the **exchangeRate.totalWei** is reduced in step 1, but this value is utilized in step 2 to calculate the **unDelegation.weightage**. This could lead to a higher value compared to the validator's accounting **weightage**, resulting in an underflow error during the unbonding process.

Consider the following scenario:

1. Alice deposits 100 BNBs into the **StakePool** and receives 100 shares in return.
   - **exchangeRate.totalWei = 0 + 100 = 100**
2. Bob deposits 200 BNBs into the StakePool and receives 200 shares in return.
   - **exchangeRate.totalWei = 100 + 200 = 300**
3. BOT delegates 300 BNBs to validator X.
   - validator.delegation.weightage = 300 * 1e4 / 300 = 1e4
4. Bob decides to withdraw all of his 200 BNBs.
   - STEP-1: Bob transfers 200 stkBNBs to the **StakePool** contract to burn.
     - **exchangeRate.totalWei = 300 - 200 = 100**
   - STEP-2: The **unbondingInitiated()** function is triggered by the BOT with **bnbUnbondValues_ = [200]**.
     - **Line 853: unDelegation.weightage = 200 * 1e4 / 100 = 2e4**
     - Here, we observe that **unDelegation.weightage** is greater than the **weightage** of validator X (**2e4 > 1e4**), resulting in a revert at line 141 in the ValidatorSet contract due to the underflow error.

```
  ValidatorSet.Info storage validator =
_validatorStore.validators[operators_[i]];
  ValidatorSet.DelegationInfo memory unDelegation = ValidatorSet.DelegationInfo({
      stakes: bnbUnbondValues_[i],
      shares: sharesToUnbond[i],
      weightage: (bnbUnbondValues_[i] * WEIGTHAGE_RATE_BASE) /
exchangeRate.totalWei
  });


  validator._undelegate(unDelegation, getTotalValidators());
```

```
    if (self.delegation.weightage - newUndelegation.weightage >
self.delegation.weightage) {
        revert InvalidParam(
            "self.delegation.weightage - newUndelegation.weightage >
self.delegation.weightage",
            newUndelegation.weightage
        );
    }
```

PRST2-8

UNABLE TO EXECUTE STAKEPOOL.EPOCHUPDATE() BECAUSE OF THE MISMATCH BETWEEN SHARES AND STAKES OF A VALIDATOR

SEVERITY: Medium

PATH:

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L675-676

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L1145-L1156

8c6ddb20774331545b6c70453eb1edca053e43af/contracts/StakePool.sol:L1117-L1119

REMEDIATION:

Consider recalculating the stakes corresponding to the calculated shares when delegating, undelegating, or redelegating the BNBs.

STATUS: Fixed

DESCRIPTION:

The function **StakePool.initiateDelegation()** is utilized to delegate the deposited BNBs from the contract to validators. Within this function, the variable **delegation.shares**, which represents the quantity of minted shares when delegating **bnbAmounts_[i]** to a validator, is calculated on line 676 by obtaining the return value of the external call **IStakeCredit(stCred).getSharesByPooledBNB(_bnbAmount)**.

```
function getSharesByPooledBNB(uint256 bnbAmount) public view returns (uint256) {
    if (totalPooledBNB == 0) revert ZeroTotalPooledBNB();
    return (bnbAmount * totalSupply()) / totalPooledBNB;
}
```

The problem arises when the aforementioned function uses a round-down calculation. This implies that when burning the **delegation.shares** immediately after minting, the **StakePool** might receive an amount of BNBs less than **bnbAmounts_[i]**. Put into a formulaic expression:

```
getPooledBNBByShares(delegation.shares) <= bnbAmounts_[i]
```

Consequently, a situation arises where the function **StakePool.epochUpdate() -> _getDailyRewards()** encounters a revert with an underflow error in lines 117 - 119 because **IStakeCredit(validator.stCred).getPooledBNB(_addressStore.getDelegationManager()) < validator.delegation.stakes**.

Consider the following scenario:

1. The StakePool contract integrates with 2 validators, X and Y, and has already delegated some BNBs to them.
2. Assume the state of validator X is:
   - **totalPooledBNB = 140**
   - **totalSupply = 100**
3. The function **StakePool.initiateDelegation()** is called to delegate 30 BNBs from the StakePool contract to validator X.
   - **Line 675: delegation.stakes = 30**
   - **Line 676: delegation.shares = 100 * 30 / 140 = 21**
   - The state of validator X becomes:
     - **totalPooledBNB = 140 + 30 = 170**
     - **totalSupply = 100 + 21 = 121**
4. Validator Y generates profit and requires the StakePool to update the **totalWei** of the StakePool by calling **epochUpdate()**. When the loop iteration reaches validator X, we have:

- Lines 117 - 119:

```
validatorReward = getPooledBNB(shares) - stakes`
                = getPooledBNB(21) - 30
                = 21 * 170 / 121 - 30
                = 29 - 30
                = -1
```

--> Reverting here due to the underflow error.

The consequence in this situation is that the **totalWei** cannot be updated until either validator X generates profit or the fund is redelegated.

Note that this rounding issues also affects the **redelegate()** process

```
stakes: bnbAmounts_[i],
shares: _getValidatorShares(operators_[i], bnbAmounts_[i]),
```

```
function _getValidatorShares(
    address _operator,
    uint256 _bnbAmount
) internal view returns (uint256) {
    if
(_validatorStore.validators[_operator]._isNotValidator(getTotalValidators()))
        revert ValidatorDoesNotExists();
    // Validator Credit Contract Address
    address stCred = _validatorStore.validators[_operator].stCred;
    uint256 shares = IStakeCredit(stCred).getSharesByPooledBNB(_bnbAmount);


    return shares;
}
```

```
uint256 validatorReward = IStakeCredit(validator.stCred).getPooledBNB(
    _addressStore.getDelegationManager()
) - validator.delegation.stakes;
```

PRST2-5

SELFDESTRUCT IS DEPRECATED AFTER THE KEPLER HARDFORK

SEVERITY: Low

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/StakedBNBToken.sol:L170-L177

REMEDIATION:

Consider removing this function because contract destruction is no longer possible.

STATUS: Fixed

DESCRIPTION:

After the BSC Kepler hardfork, the StakedBNBToken.sol contract utilizes the selfdestruct opcode, which was deprecated. Post-hardfork, this opcode solely sends BNB to msg.sender without initiating contract destruction.

```
function selfDestruct()
        external
        override
        onlySender(_addressStore.getTimelockedAdmin())
        whenPaused
{
        selfdestruct(payable(_owner));
}
```

PRST2-9

UNNECESSARY IF STATEMENT IN THE DELEGATIONMANAGER.DELEGATEDEPOSITEDBNB() FUNCTION

SEVERITY: Low

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/DelegationManager.sol:L126

REMEDIATION:

Consider removing the if statement in line 126.

STATUS: Fixed

DESCRIPTION:

In the **DelegationManager.delegateDepositedBNB()** function, there exists a validation on line 126 to ensure that **msg.value** is not less than the available amount of BNBs in the **StakePool** contract. However, this validation is redundant. This redundancy arises from the fact that in the **StakePool.initiateDelegation()** function, all the **excessBNB** are forwarded to the **DelegationManager** contract. Consequently, when the call **IStakePoolBot(stakePool).getDeposits()** is made on line 126 of the **DelegationManager** contract, it will inevitably return a value of 0 (because **StakePool.balance == _claimReserve**). Given that **msg.value** is always greater than or equal to 0, this validation will always return true.

```
if (msg.value >= IStakePoolBot(stakePool).getDeposits()) {
```

PRST2-11

LACK OF TESTS

SEVERITY: Low

REMEDIATION:

Consider adding tests to cover all execution branches to prevent unintended behavior.

STATUS: Fixed

DESCRIPTION:

A large portion of sensitive contract functionality is not covered by tests.

This leads to several potential issues:

1. The correctness of the code can only be assessed based on partial and changing documentation. This is because intended and unintended behaviors are not captured in tests.
2. Introduction of new vulnerabilities for established code in future code changes, since known positive and negative behaviors are not checked automatically.
3. Higher likelihood of missed vulnerabilities in current and future development and review.

PRST2-12

DEPOSITS CAN TEMPORARILY STUCK IN DELEGATIONMANAGER CONTRACT

SEVERITY: Low

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/StakePool.sol:L697-L733

REMEDIATION:

Consider to send as msg.value only the sum of the values from bnbAmounts_ array to prevent such behavior.

STATUS: Fixed

DESCRIPTION:

During the delegation process in function **initiateDelegation**, Bot always passes **excessBNB** variable as **msg.value** to the **DelegationManager** contract, even when **excessBNB** is bigger than the sum of the values in **bnbAmounts_** argument. This behavior can lead to the problems when Bot does not take into account one of the deposits during the delegation process.

Considering next scenario:

1. User1 deposits 1 BNB into **StakePool**.
2. User2 deposits 1 BNB into **StakePool**.
3. While handling only the first deposit, Bot calls **initiateDelegation** function with **bnbAmounts** = 1 BNB, but with **msg.value** = 2 BNB.
4. Only 1 BNB was delegated to the operator and 1 BNB remains on the **DelegationManager** contract.

This 1 BNB can be returned to the **StakePool** contract via function **claimUnbondedBNB** or via second call to **initiateDelegation** function with one additional delegation (because of **excessBNB > config.minDelegationAmountcheck**).

```
function initiateDelegation(
    address[] calldata operators_,
    uint256[] calldata bnbAmounts_
) external override whenNotPaused onlyRole(BOT_ROLE) {
    if (operators_.length != bnbAmounts_.length) {
        revert ArgumentsLengthMismatch();
    }
    // contract will always have at least the _claimReserve, so this should never
overflow.
    uint256 excessBNB = getDeposits();



    // Initiate a delegate only if deposited BNB > 1 BNB
    if (excessBNB > config.minDelegationAmount) {
        // Note that the probability of a black swan event is very low. On top of
that, as time passes, we will be
        // accumulating some stkBNB as rewards in FeeVault. This implies that our
share of BNB in the pool will
         // keep increasing over time. As long as this share is more than the
total fee spent till that time, we need
          // not worry about paying back the fee losses. Also, for us to be
economically successful, we must set
         // protocol fee rates in a way so that the rewards we earn via FeeVault
are significantly more than the fee
        // we are paying for the protocol operations.
                                                    bool       delegated       =
IDelegationManager(payable(_addressStore.getDelegationManager()))
            .delegateDepositedBNB{ value: excessBNB }(operators_, bnbAmounts_);



        if (!delegated) {
            revert DepositsDelegationFailed(excessBNB);
        }
```

```
        for (uint256 i = 0; i < operators_.length; ++i) {
                                        ValidatorSet.Info   storage   validator   =
_validatorStore.validators[operators_[i]];


            uint256 newStake = IStakeCredit(validator.stCred).getPooledBNB(
                _addressStore.getDelegationManager()
            );


            validator._delegate(newStake, block.timestamp);
        }


        emit InitiateDelegation_Transfered(excessBNB);
    }
```

PRST2-18

FUNCTION _ISACTIVEVALIDATOR() DOES NOT CHECK THE STATE OF STAKEHUB CONTRACT

SEVERITY: Low

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/embedded-libs/ValidatorSet.sol:L153

REMEDIATION:

Consider using getValidatorBasicInfo from StakeHub contract.

STATUS: Fixed

DESCRIPTION:

Function **_isActiveValidator()** is checking only the internal state of **StakePool** contract, which can be updated only by bot. It may lead to situations when validator is jailed in **StakeHub** contract, but has an **Active** status in **StakePool** contract.

```
    function _isActiveValidator(Info memory self) internal pure returns
(bool) {
        return self.status == Status.Active;
    }
```

PRST2-19

VALIDATOR STATUS IS NOT CHECKED IN THE INITIATEDELEGATION() FUNCTION

SEVERITY: Low

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/StakePool.sol:L697

STATUS: Fixed

DESCRIPTION:

The status of validator is not checked during the delegation process, so it is possible to delegate BNB to the disabled validator.

```
function initiateDelegation(
        address[] calldata operators_,
        uint256[] calldata bnbAmounts_
    ) external override whenNotPaused onlyRole(BOT_ROLE) {
        if (operators_.length != bnbAmounts_.length) {
            revert ArgumentsLengthMismatch();
        }
         // contract will always have at least the _claimReserve, so this should never overflow.
        uint256 excessBNB = getDeposits();

        // Initiate a delegate only if deposited BNB > 1 BNB
        if (excessBNB > config.minDelegationAmount) {
             // Note that the probability of a black swan event is very low. On top of that, as time passes, we will be
            // accumulating some stkBNB as rewards in FeeVault. This implies that our share of BNB in the pool will
            // keep increasing over time. As long as this share is more than the total fee spent till that time, we need
             // not worry about paying back the fee losses. Also, for us to be economically successful, we must set
              // protocol fee rates in a way so that the rewards we earn via FeeVault are significantly more than the fee
            // we are paying for the protocol operations.
                                                    bool        delegated        = IDelegationManager(payable(_addressStore.getDelegationManager()))
                            .delegateDepositedBNB{  value:  excessBNB  }(operators_, bnbAmounts_);

            if (!delegated) {
                revert DepositsDelegationFailed(excessBNB);
            }

            for (uint256 i = 0; i < operators_.length; ++i) {
                                        ValidatorSet.Info    storage    validator    = _validatorStore.validators[operators_[i]];

                uint256 newStake = IStakeCredit(validator.stCred).getPooledBNB(
                    _addressStore.getDelegationManager()
                );
```

```
                validator._delegate(newStake, block.timestamp);
            }

            emit InitiateDelegation_Transfered(excessBNB);
        } else if (excessBNB > 0 && _bnbToUnbond > 0) {
            // if the excess amount is so small that it can't be moved to BBC and
there is _bnbToUnbond, short-circuit
             // the bot process and directly update the _claimReserve. This way,
we will still be able to satisfy claims
             // even if the total deposit throughout the unstaking epoch is less
than the minimum cross-chain transfer.

             // The reason we don't do this short-circuit process more generally
is because the short-circuited amount
            // doesn't earn any rewards. While, if it would have gone through the
normal transferOut process, it would
            // have earned significant staking rewards because we undelegate only
once in 7 days on mainnet. So, we do
             // this short-circuit process only to handle the edge case when the
deposits throughout the unstaking epoch
            // are less than the minimum cross-chain transfer and users initiated
withdrawals, so that we are
            // successfully able to satisfy the claim requests.

            uint256 shortCircuitAmount;
            if (_bnbToUnbond > excessBNB.toInt256()) {
                 // all the excessBNB we have in the contract will be used up to
satisfy claims. The remaining
                // _bnbToUnbond will be made available to the contract by the bot
via the unstaking operations.
                shortCircuitAmount = excessBNB;
            } else {
                   // the amount we need to unbond to satisfy claims is already
present in the contract. So, only that much
                // needs to be moved to _claimReserve.
                shortCircuitAmount = uint256(_bnbToUnbond);
            }
            _bnbToUnbond -= shortCircuitAmount.toInt256();
            _claimReserve += shortCircuitAmount;

            emit InitiateDelegation_ShortCircuit(shortCircuitAmount);
        }
```

```
        // else there is no excess amount or very small excess amount and no
_bnbToUnbond. In these cases, the excess
        // amount will remain in the contract, and will be delegated the next
day.

       // emitted to make the life of off-chain dependencies easy
       emit InitiateDelegation_Success();
    }
```

PRST2-10

REDUNDANT CONDITIONS ARE PRESENT IN THE CHECK FUNCTION OF THE VALIDATORSET CONTRACT

SEVERITY: Informational

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/embedded-libs/ValidatorSet.sol:L98-L103

REMEDIATION:

Consider removing these requirements.

STATUS: Fixed

DESCRIPTION:

In lines 98 to 103, within the function **ValidatorSet._checkDelegate()**, there's a validation:

```
if (self.delegation.stakes + newDelegation.stakes < self.delegation.stakes)
{
    revert InvalidParam(
        "self.delegation.stakes + newDelegation.stakes <
self.delegation.stakes",
        newDelegation.stakes
    );
}
```

This requirement is unnecessary since all variables involved in this condition are greater than or equal to 0. Hence, the left side will always be greater than or equal to the right side.

PRST2-17

REDUNDANT IMPORTS

SEVERITY: Informational

PATH:

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/StakePool.sol:L19

524d33878fb37bb584516199bb6d997c5536c7c0/contracts/DelegationManager.sol:L11

REMEDIATION:

Consider deleting hardhat console imports.

STATUS: Fixed

DESCRIPTION:

Library **hardhat/console.sol** is imported in several contracts. This library is used for testing purposes. It is recommended to delete this dependency in production environment.

```
import "hardhat/console.sol";
```


hexens
×
PERSISTENCE