// HALBORN

Persistence - stkBNB

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **July 10th, 2022 - August 3nd, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/27/2022	Luis Buendia
0.2	Draft Review	07/29/2022	Gabi Urrutia
1.0	Remediation Plan	08/04/2022	Luis Buendia
1.1	Remediation Plan Review	08/05/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Luis Buendia	Halborn	Luis.Buendia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Persistence engaged Halborn to conduct a security audit on their smart contracts beginning on July 10th, 2022 and ending on August 3nd, 2022 . The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that were mostly addressed by the Persistence team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the contracts' solidity code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts. (Hardhat).
- Static Analysis of security for scoped contract, and imported functions manually.
- Testnet deployment (Ganache).

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the following smart contracts on the prepared branch for the audit:

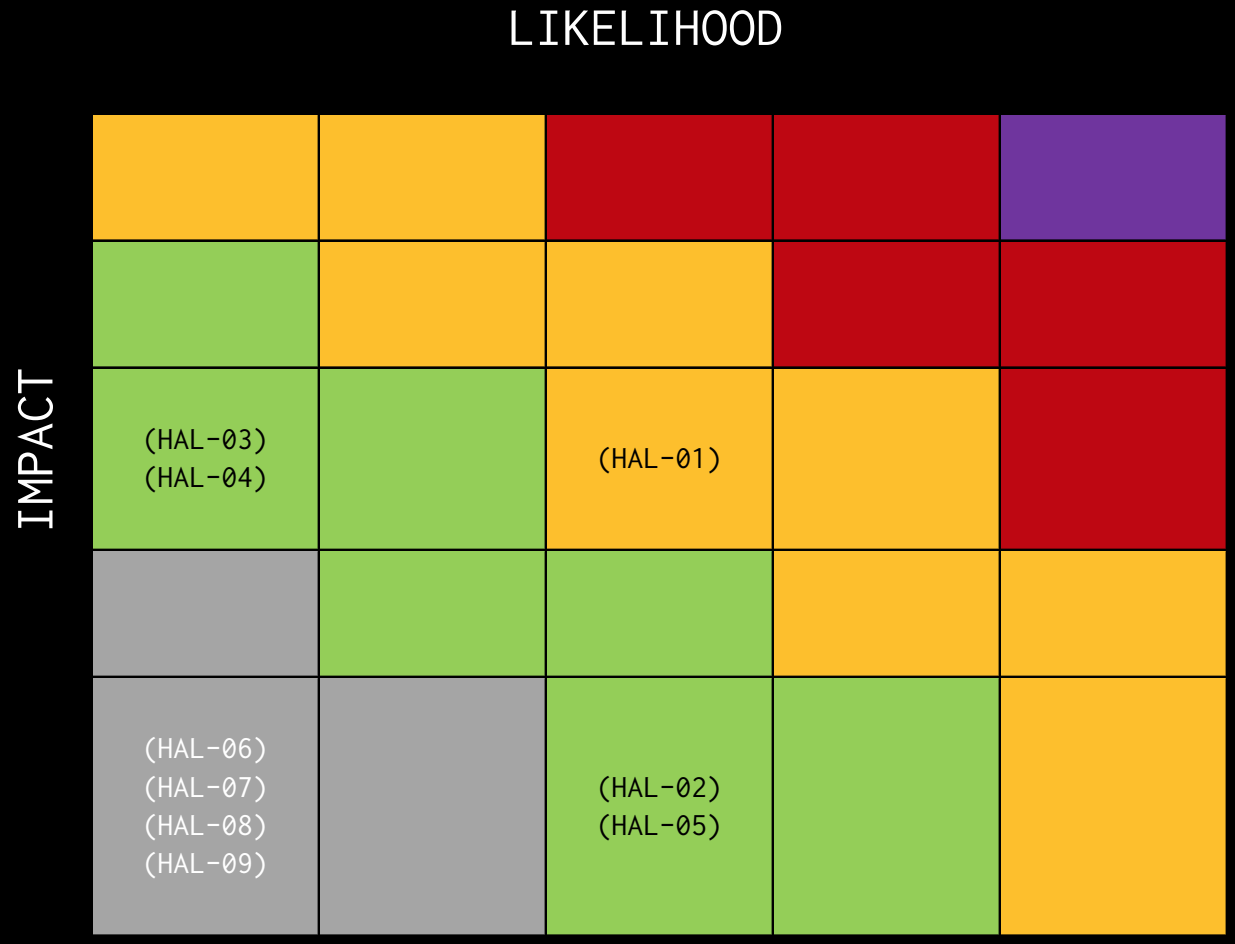
Commit ID: bde7ee900aba18aecd0e8e0c0497121540dd5abb.

FIX Commit TREE/ID :

Main Branch
Commit ID

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	4	4



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - USAGE OF SELF-DESTRUCT MAY LEAD TO FUNDS LOSS	Medium	SOLVED - 08/02/2022
HAL-02 - IMPROPER CHECK ON CLAIM FUNCTION	Low	SOLVED - 08/02/2022
HAL-03 - UNSAFE CASTING	Low	SOLVED - 08/02/2022
HAL-04 - FEE VAULT SHOULD NOT BE ABLE TO SEND TOKENS TO STAKEPOOL CONTRACT	Low	SOLVED - 08/02/2022
HAL-05 - USE CALL INSTEAD OF SEND OR TRANSFER	Low	SOLVED - 08/02/2022
HAL-06 - MISSING TWO-STEP TRANSFER OWNERSHIP PATTERN	Informational	SOLVED - 08/02/2022
HAL-07 - STORING VARIABLE CAN SAVE GAS	Informational	SOLVED - 08/02/2022
HAL-08 - ADDING UNCHECKED DIRECTIVE CAN SAVE GAS	Informational	ACKNOWLEDGED
HAL-09 - PREFIX INCREMENTS ARE CHEAPER THAN POSTFIX INCREMENTS	Informational	SOLVED - 08/02/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) USAGE OF SELF-DESTRUCT MAY LEAD TO FUNDS LOSS - MEDIUM

Description:

The usage of the selfdestruct function erases the contract code from the blockchain. Although it is understandable under certain circumstances, this function if executed inappropriately can lead to the destruction of the token contract making users loose all the deposited money and making the decentralized application broken.

Code Location:

Listing 1: StakeBNBToken.sol

```
125 function selfDestruct(address addr) external onlyRole(
 ↳ DEFAULT_ADMIN_ROLE) whenPaused {
126     selfdestruct(payable(addr));
127     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Do not implement the self-destruct without checking user's funds has been given back.

Remediation Plan:

SOLVED: The persistenceOne team fixed the above issue by using the timelockcontract from openzeppelin on the commit ID d059bccbb368158a63767107b37894d47009c385. As a result, this gives time

to the users to call the withdraw function in case the self-destruct function is triggered.

Moreover, the persistenceOne team ensures that this function will only be called in case a major upgrade occurs on the Binance ecosystem and gets the compromise to re-balance the stkBNB tokens to every user account if this situation happens.

3.2 (HAL-02) IMPROPER CHECK ON CLAIM FUNCTION - LOW

Description:

The **_claim** function checks the amount to withdraw from the protocol is less than the current contract balance. However, this check is not correct, and it can lead to revert the transaction as the claimReserve can be 0, although the contract balance is not 0. So on the subtraction on L786, the contract reverts with an unhandled arithmetic exception.

Code Location:

Listing 2: StakePool.sol

```
781 if (address(this).balance < req.weiToReturn) {
782     revert InsufficientFundsToSatisfyClaim();
783 }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

Check that the amount to be returned is less than the **claimReserve**.

Remediation Plan:

SOLVED: The persistenceOne team fixed the above issue on the commit ID d059bccbb368158a63767107b37894d47009c385.

3.3 (HAL-03) UNSAFE CASTING - LOW

Description:

The **StakePool** contracts performs unsafe casting from uint256 to int256 that can lead to overflow. Although the risk is minimum as the amount of BNB is still far from reaching those numbers.

Code Location:

Listing 3: StakePool.sol

```
560 if (_bnbToUnbond > int256(excessBNB)) {
```

Listing 4: StakePool.sol

```
569 _bnbToUnbond -= int256(shortCircuitAmount);
```

Listing 5: StakePool.sol

```
621 _bnbToUnbond -= int256(bnbUnbonding_);
```

Listing 6: StakePool.sol

```
744 _bnbToUnbond += int256(weiToReturn);
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

To safely avoid the unlikely situation, it is recommended to use the **SafeCast** library from OpenZeppelin.

Remediation Plan:

SOLVED: The persistenceOne team fixed the above issue on the commit ID d059bccbb368158a63767107b37894d47009c385. The final implementation contains the recommended library to avoid overflows on arithmetic operations when casting.

3.4 (HAL-04) FEE VAULT SHOULD NOT BE ABLE TO SEND TOKENS TO STAKEPOOL CONTRACT - LOW

Description:

The **claimStkBNB** function from the **FeeVault** contract can send the fees to the StakePool contract. Although this action is restricted by the **onlyOwner** modifier, this action may lead the fees to be locked on the **StakePool** contract forever. This happens because there is no way to call the claim function from the **FeeVault** contract.

Code Location:

Listing 7: FeeVault.sol

```
75 function claimStkBNB(address recipient, uint256 amount) external
↳ override onlyOwner {
76         IStakedBNBToken(addressStore.getStkBNB()).send(recipient,
↳ amount, "");
77
78         emit Withdraw(msg.sender, recipient, amount);
79     }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Ensure that the recipient address is not the **stakepool** contract to avoid a possible situation where funds maybe locked in the contract.

Remediation Plan:

SOLVED: The persistenceOne team fixed the above issue on the commit ID d059bccbb368158a63767107b37894d47009c385.

3.5 (HAL-05) USE CALL INSTEAD OF SEND OR TRANSFER - LOW

Description:

The usage of send or transfer limits the amount of gas send in the transaction to 2300. This can be considered a safeguard to avoid reentrancy. However, given the current protections of the contracts, it does seem feasible to use call function without further risk. This implies that the system expects to be used by some contracts that can execute more complex code on the fallback function.

Code Location:

Listing 8: StakePool.sol (Line 793)

```
770     function _claim(uint256 index) internal returns (bool) {
771         if (index >= claimReqs[msg.sender].length) {
772             revert IndexOutOfBounds(index);
773         }
774
775         // find the requested claim
776         ClaimRequest memory req = claimReqs[msg.sender][index];
777
778         if (!_canBeClaimed(req)) {
779             return false;
780         }
781         if (_claimReserve < req.weiToReturn) {
782             revert InsufficientFundsToSatisfyClaim();
783         }
784
785         // update _claimReserve
786         _claimReserve -= req.weiToReturn;
787
788         // delete the req, as it has been fulfilled (swap deletion
 ↳  for O(1) compute)
789         claimReqs[msg.sender][index] = claimReqs[msg.sender][
 ↳ claimReqs[msg.sender].length - 1];
790         claimReqs[msg.sender].pop();
791
```

```
792         // return BNB back to user
793         payable(msg.sender).transfer(req.weiToReturn);
794         emit Claim(msg.sender, req, block.timestamp);
795         return true;
796     }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

If the Persistence team consider it appropriate, change the transfer function to call function to allow contracts to execute code in the fallback function.

Remediation Plan:

SOLVED: The persistenceOne team fixed the above issue on the commit ID d059bccbb368158a63767107b37894d47009c385.

3.6 (HAL-06) MISSING TWO-STEP TRANSFER OWNERSHIP PATTERN - INFORMATIONAL

Description:

The **AddressStore** contract use **Ownable** from **OpenZeppelin** which is a simple mechanism to transfer the ownership not supporting a two-step transfer ownership pattern. **Ownable** is a simpler mechanism with a single owner "role" that can be assigned to a single account. This simpler mechanism can be useful for quick tests, but projects with production concerns are likely to outgrow it. Transferring ownership is a critical operation and this could lead to transferring it to an inaccessible wallet or renouncing the ownership, e.g. mistakenly.

Code Location:

AddressStore Contract

Listing 9: AddressStore.sol

```
8 contract AddressStore is IAddressStore, Ownable
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to implement a two-step transfer of ownership mechanism where the ownership is transferred and later claimed by a new owner to confirm the whole process and prevent lockout. As OpenZeppelin ecosystem does not provide such implementation, it has to be done in-house. For the

inspiration BoringOwnable can be considered, however it has to be well tested, especially if it is integrated with other OpenZeppelin contracts used by the project.

References

- Access Control
- BoringOwnable.sol

Remediation Plan:

SOLVED: The persistenceOne team does not require a fix for this, as the platform will use a multi-signature wallet when developed on production.

3.7 (HAL-07) STORING VARIABLE CAN SAVE GAS - INFORMATIONAL

Description:

Calling a contract, although a view function, still requires some gas from the contract. The **UndelegationHolder** contract performs the call to the **StakePool** contract twice instead of storing the result on a variable.

Risk Level:

Likelihood - 1
Impact - 1

Code Location:

UndelegationHolder contract

Listing 10: UndelegationHolder.sol

```
66 if (amountToSend > IStakePoolBot(stakePool).bnbUnbonding()) {
67     amountToSend = IStakePoolBot(stakePool).bnbUnbonding();
68 }
```

Recommendation:

Store the result of the function call in a variable and use the stored value in subsequent operations.

Remediation Plan:

SOLVED: The persistenceOne team fixed the above issue on the commit ID d059bccbb368158a63767107b37894d47009c385.

3.8 (HAL-08) ADDING UNCHECKED DIRECTIVE CAN SAVE GAS - INFORMATIONAL

Description:

For the arithmetic operations that will never over/underflow, using the **unchecked** directive (Solidity v0.8 has default overflow/underflow checks) can save some gas from the unnecessary internal over/underflow checks.

Risk Level:

Likelihood - 1
Impact - 1

Code Location:

StakePool contract

Listing 11: StakePool.sol

```
539 _TOKEN_HUB.transferOut{ value: excessBNB }(
540     _ZERO_ADDR,
541     config.bcStakingWallet,
542     transferOutAmount,
543     uint64(block.timestamp + 3600)
544 );
```

Recommendation:

Using the unchecked keyword to avoid redundant arithmetic checks and save gas when an underflow/overflow cannot happen.

Remediation Plan:

ACKNOWLEDGED: The persistenceOne team acknowledged this issue.

3.9 (HAL-09) PREFIX INCREMENTS ARE CHEAPER THAN POSTFIX INCREMENTS - INFORMATIONAL

Description:

The function `getPaginatedClaimRequests` uses `i++` which costs more gas than `++i`, especially in a loop.
In the loops below, postfix (e.g. `i++`) operators were used to increment or decrement variable values. It is known that, in loops, using prefix operators (e.g. `++i`) costs less gas per iteration than using postfix operators.

Risk Level:

Likelihood - 1
Impact - 1

Code Location:

StakePool Contract

Listing 12: StakePool.sol

```
722 for (uint256 i = 0; i < to - from; i++) {
723             paginatedClaimRequests[i] = claimReqs[user][from + i];
724 }
```

Recommendation:

It is recommended to use unchecked `++i` and `--j` operations instead of `i++` and `j--` to increment or decrement the values of `uint` variables within loops. This applies not only to iterator variables, but also to increments and decrements performed within loop code blocks.

Remediation Plan:

SOLVED: The persistenceOne team fixed the above issue on the commit ID d059bccbb368158a63767107b37894d47009c385.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

```
luisbuendia@Luiss-MacBook-Air persistenceOne/stkBNB-contracts (audit/halborn *%) » slither contracts/AddressStore.sol --solc-remaps @openzeppelin/=/Users/luisbuendia/projects/per
sistenceOne/stkBNB-contracts/node_modules/@openzeppelin/

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.7']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.7 (contracts/AddressStore.sol#3)
        - ^0.8.7 (contracts/interfaces/IAddressStore.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.sol#21-23) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/contracts/access/Ownable.sol#69-72)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
contracts/AddressStore.sol analyzed (4 contracts with 78 detectors), 6 result(s) found
```

```
luisbuendia@Luiss-MacBook-Air persistenceOne/stkBNB-contracts (audit/halborn *%) » slither contracts/FeeVault.sol --solc-remaps @openzeppelin/=/Users/luisbuendia/projects/persist
enceOne/stkBNB-contracts/node_modules/@openzeppelin/

Reentrancy in FeeVault.claimStkBNB(address,uint256) (contracts/FeeVault.sol#75-79):
        External calls:
        - IStakedBNBToken(addressStore.getStkBNB()).send(recipient,amount,) (contracts/FeeVault.sol#76)
        Event emitted after the call(s):
        - Withdraw(msg.sender,recipient,amount) (contracts/FeeVault.sol#78)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#174-194) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#186-189)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.2', '^0.8.7']
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC777/IERC777RecipientUpgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/introspection/IERC1820RegistryUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC777/IERC777.sol#4)
        - ^0.8.7 (contracts/FeeVault.sol#3)
        - ^0.8.7 (contracts/interfaces/IAddressStore.sol#3)
        - ^0.8.7 (contracts/interfaces/IBEP20.sol#3)
        - ^0.8.7 (contracts/interfaces/IFeeVault.sol#3)
        - ^0.8.7 (contracts/interfaces/IStakedBNBToken.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

AddressUpgradeable.functionCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#85-87) is never used and should be removed
AddressUpgradeable.functionCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#95-101) is never used and should be removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#114-120) is never used and should b
e removed
AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-139) is never used and s
hould be removed
AddressUpgradeable.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#147-149) is never used and should be removed
AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#157-166) is never used and should be re
moved
AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65) is never used and should be removed
AddressUpgradeable.verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#174-194) is never used and should be removed
```

```
luisbuendia@Luiss-MacBook-Air persistenceOne/stkBNB-contracts (audit/halborn *%) » slither contracts/StakePool.sol --solc-remaps @openzeppelin/=/Users/luisbuendia/projects/persis
tenceOne/stkBNB-contracts/node_modules/@openzeppelin/

AccessControlEnumerableUpgradeable._grantRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlEnumerableUpgradeable.sol#58-61) ignores retu
rn value by _roleMembers[role].add(account) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlEnumerableUpgradeable.sol#60)
AccessControlEnumerableUpgradeable._revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlEnumerableUpgradeable.sol#66-69) ignores ret
urn value by _roleMembers[role].remove(account) (node_modules/@openzeppelin/contracts-upgradeable/access/AccessControlEnumerableUpgradeable.sol#68)
StakePool.initiateDelegation() (contracts/StakePool.sol#519-574) ignores return value by _TOKEN_HUB.transferOut{value: excessBNB}(_ZERO_ADDR,config.bcStakingWallet,transferOutAmo
unt,uint64(block.timestamp + 3600)) (contracts/StakePool.sol#539-544)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

IStakePoolBot.unbondingInitiated(uint256).bnbUnbonding (contracts/interfaces/IStakePoolBot.sol#81) shadows:
        - IStakePoolBot.bnbUnbonding() (contracts/interfaces/IStakePoolBot.sol#37) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

StakePool._claim(uint256) (contracts/StakePool.sol#770-796) has external calls inside a loop: address(msg.sender).transfer(req.weiToReturn) (contracts/StakePool.sol#793)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
```

```
luisbuendia@Luiss-MacBook-Air persistenceOne/stkBNB-contracts (audit/halborn *%) » slither contracts/StakedBNBToken.sol --solc-remaps @openzeppelin/=/Users/luisbuendia/projects/p
ersistenceOne/stkBNB-contracts/node_modules/@openzeppelin/

StakedBNBToken.selfDestruct(address) (contracts/StakedBNBToken.sol#125-127) allows anyone to destruct the contract
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#suicidal

AccessControlEnumerable._grantRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControlEnumerable.sol#52-55) ignores return value by _roleMembers[role].add
(account) (node_modules/@openzeppelin/contracts/access/AccessControlEnumerable.sol#54)
AccessControlEnumerable._revokeRole(bytes32,address) (node_modules/@openzeppelin/contracts/access/AccessControlEnumerable.sol#60-63) ignores return value by _roleMembers[role].re
move(account) (node_modules/@openzeppelin/contracts/access/AccessControlEnumerable.sol#62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Reentrancy in ERC777._burn(address,uint256,bytes,bytes) (node_modules/@openzeppelin/contracts/token/ERC777/ERC777.sol#390-414):
        External calls:
        - _callTokensToSend(operator,from,address(0),amount,data,operatorData) (node_modules/@openzeppelin/contracts/token/ERC777/ERC777.sol#400)
                - IERC777Sender(implementer).tokensToSend(operator,from,to,amount,userData,operatorData) (node_modules/@openzeppelin/contracts/token/ERC777/ERC777.sol#473)
        State variables written after the call(s):
        - _balances[from] = fromBalance - amount (node_modules/@openzeppelin/contracts/token/ERC777/ERC777.sol#408)
        - _totalSupply -= amount (node_modules/@openzeppelin/contracts/token/ERC777/ERC777.sol#410)
Reentrancy in ERC777._send(address,address,uint256,bytes,bytes,bool) (node_modules/@openzeppelin/contracts/token/ERC777/ERC777.sol#363-381):
```

```
luisbuendia@Luiss-MacBook-Air persistenceOne/stkBNB-contracts (audit/halborn *%) » slither contracts/UndelegationHolder.sol --solc-remaps @openzeppelin/=/Users/luisbuendia/projec
ts/persistenceOne/stkBNB-contracts/node_modules/@openzeppelin/

IStakePoolBot.unbondingInitiated(uint256).bnbUnbonding (contracts/interfaces/IStakePoolBot.sol#81) shadows:
        - IStakePoolBot.bnbUnbonding() (contracts/interfaces/IStakePoolBot.sol#37) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Low level call in UndelegationHolder.withdrawUnbondedBNB() (contracts/UndelegationHolder.sol#52-80):
        - (sent) = stakePool.call{value: amountToSend}() (contracts/UndelegationHolder.sol#71-74)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
contracts/UndelegationHolder.sol analyzed (4 contracts with 78 detectors), 2 result(s) found
```

As a result of the tests completed with the Slither tool, some results were obtained and these results were reviewed by Halborn. In line with the reviewed results, it was decided that some vulnerabilities were false-positive and these results were not included in the report. The actual vulnerabilities found by Slither are already included in the findings on the report.

THANK YOU FOR CHOOSING

// HALBORN