BAILSEC.IO

OFFICE@BAILSEC.IO

X: @BAILSECURITY

TG: @HELLOATBAILSEC

FINAL REPORT:

Stader Labs bnbX

July 2024

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Important:
Please ensure that the deployed contract matches the source-code of the last commit hash.

Project	Stader Labs - bnbX
Website	staderlabs.com
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/stader-labs/bnbX/blob/2bba18c23cf0b2fdf03e80be7cd0976479bb3d1d/contracts/StakeManagerV2.sol
Resolution 1	https://github.com/stader-labs/bnbX/blob/1445cb1917ace80d11627ff85a8815b2e8465b53/contracts/StakeManagerV2.sol
Resolution 2	https://github.com/stader-labs/bnbX/blob/51b09e626dfc2acd7ee2a5d2fd6936f35731c70a/contracts/StakeManagerV2.sol
Resolution 3	https://github.com/stader-labs/bnbX/tree/aee951c5477fa8091f25386980fddeac905f9e20/contracts

2. Detection Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	4	4		
Medium	4	3		1
Low	3	2		1
Informational	3	1		2
Governance	1			1
Total	15	10		5

2.1 Detection Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior

2. Detection

OperatorRegistry

The OperatorRegistry is a simple registry contract that keeps track of all involved operator addresses, storing them in an enumerableSet. An operator address is corresponding to a validator in the StakeHub contract and will be used to delegate BNB towards it.

Any address with the DEFAULT_ADMIN_ROLE can add, remove and set a preferred deposit/withdrawal operator.

This contract employs OpenZeppelin's AccessControlUpgradeable library for access control purposes.

Privileged Functions:

- grantRole
- revokeRole
- setPreferredDepositOperator
- setPreferredWithdrawalOperator
- togglePause

Issue_13	Usage of floating pragma is discouraged
Severity	**Informational**
Description	A strict pragma specifies a particular compiler version to compile the smart contract. This ensures that the contract is always compiled with the same version, avoiding any unexpected behaviors or bugs introduced in newer compiler releases. Compiler updates can sometimes introduce changes that alter the way contracts are interpreted or executed, potentially opening up security vulnerabilities if not adequately tested.
Recommendations	Consider using a strict pragma version.
Comments / Resolution	Resolved.

StakeManagerV2

Bailsec was tasked with a re-audit of the StakeManagerV2 because the contract was refactored, the commit to audit is the following:

https://github.com/stader-labs/bnbX/blob/2bba18c23cf0b2fdf03e80be7cd0976479bb3d1d/contracts/StakeManagerV2.sol

This will be a completely new audit and will disregard all information from the past audit.

The StakeManagerV2 is the entry contract which allows users to provide BNB and receive BNBx. The received BNBx amount is determined by the **rule of three**, which is widely known from vault implementations:

*bnbAmount * totalShares / totalDelegated*

In that calculation these parameters are defined as follows:

- bnbAmount: The provided BNB amount by the staker
- totalShares: Existing supply of BNBx
- totalDelegated: Amount of BNB delegated to operators (since last update)

This means that users will receive a BNBx amount based on their pro-rata contribution on the overall BNB in the system. The delegation can happen to a pool of operators whereas it will always delegate it to the preferred operator as determined within the OperatorRegistry. This can however be changed by governance.

The main purpose for users to stake their BNB is the price appreciation of BNBx, which is mainly driven by staking rewards but can also be incentivized through the owner via the delegateWithoutMinting function. The mechanism which is responsible for the price appreciation is the increase of the amount of pooled BNB through all operators. There are two notable things to mention about the price appreciation:

a) It is not handled automatically but only manually whenever the updateER function is triggered
b) A part of the price-appreciation is taken as fee and minted to the Stader Treasury as BNBx

Users can burn their BNBx and receive BNB through a withdrawal queue. We will quickly explain the flow:

a) Alice invokes the requestWithdraw function with the amount of BNBx she wants to redeem for BNB. This will push a new withdrawalRequest into the withdrawalRequests array.
b) Governance invokes the startBatchUndelegation function which will subsequently fetch elements in the withdrawalRequest and creates a new batchWithdrawalRequest that aggregates a certain amount of requests (determined by _batchSize). The sum of these requests is then undelegated from the STAKE_HUB and will be claimable 7 days later.
c) After 7 days, anyone can invoke the completeBatchUndelegation function which claims the latest batch request from the STAKE_HUB. This marks the point where users can start claiming their withdraw requests.
d) Users can now claim their requests (which are allocated to the claimed batch request) via the claimWithdrawal function.

Privileged Functions:

- startBatchUndelegation
- redelegate
- forceUpdateER
- delegateWithoutMinting
- pause
- unpause
- setStaderTreasury
- setFeeBps
- setMaxActiveRequestsPerUser
- setMaxExchangeRateSlippageBps

Issue_01	Governance Privilege: Users are completely dependent on the operator to withdraw funds
Severity	**Governance**
Description	To withdraw funds, users must first call the contract to initiate a request. Then, the operator calls the contract to process the user's request. If the operator does not invoke the startBatchUndelegation function, users will be unable to withdraw their funds. Moreover, the contract is under an upgradeable proxy which can result in a total loss of funds if the proxy admin key is compromised.
Recommendations	Consider incorporating a Gnosis Multisignature contract as owner and ensuring that the Gnosis participants are trusted entities.
Comments / Resolution	Acknowledged.

Issue_02	Blunder within claimWIthdrawal allows users to immediately claim requested withdrawals
Severity	**High**
Description	The claimWithdrawal function allows users to claim their fulfilled requests. It executes a check if the request has already been claimed and fetches the corresponding batchWithdrawalRequest. The problem is that requests will initially always point to batchId = 0 upon request creation and the batchId = 0 is actually a valid created batch request (it is the very first created batch request): https://github.com/stader-labs/bnbX/blob/2bba18c23cf0b2fdf03e80be7cd0976479bb3d1d/con

	tracts/StakeManagerV2.sol#L115 Therefore, the isClaimable check will never revert and users can claim their unprocessed requests. This will result in several down-stream issues.
Recommendations	Consider checking if the withdrawalRequest has actually been processed: https://github.com/stader-labs/bnbX/blob/2bba18c23cf0b2fdf03e80be7cd0976479bb3d1d/contracts/StakeManagerV2.sol#L373
Comments / Resolution	Resolved, two changes have been implemented: > default batchId is now uint256.max > processing is checked upon claimWithdrawal

Issue_03	`startBatchUndelegation` fails when processing the last element of the `withdrawalRequests` queue
Severity	**High**
Description	The `startBatchUndelegation` function calls `_computeBnbXToBurn` to calculate the amount of shares that need to be undelegated. The `_computeBnbXToBurn` function iterates through the `withdrawalRequests` queue until the following condition is not met. ```solidity while ((processedCount < _batchSize) && (firstUnprocessedUserIndex < withdrawalRequests.length) && (cummulativeBnbToWithdraw <= pooledBnb)) { amountInBnbXToBurn = cummulativeBnbXToBurn;

	withdrawalRequests[firstUnprocessedUserIndex].processed = true; withdrawalRequests[firstUnprocessedUserIndex].batchId = batchWithdrawalRequests.length; processedCount++; firstUnprocessedUserIndex++; // below line won't end up in infinite loop, these checks will stop it // (processedCount < _batchSize) && (firstUnprocessedUserIndex < withdrawalRequests.length) cummulativeBnbXToBurn += withdrawalRequests[firstUnprocessedUserIndex].amountInBnbX; cummulativeBnbToWithdraw = convertBnbXToBnb(cummulativeBnbXToBurn); } ``` When `firstUnprocessedUserIndex` is `withdrawalRequests.length-1`, the last member of `withdrawalRequests` will be processed in the loop. However, `firstUnprocessedUserIndex` will be accumulated again in the loop and used as an index to access `withdrawalRequests`. This will fail due to array out-of-bounds access. This results in a failure if the withdrawal batch contains the last member of `withdrawalRequests`.
Recommendations	If `firstUnprocessedUserIndex >= withdrawalRequests.length`, use `break` to end the loop.
Comments / Resolution	Resolved, the logic has been refactored and is now as follows: > enter the while loop as long as we are below _batchSize and still have outstanding requests > aggregate desired BNBx amount from request > convert it to corresponding BNB amount > break if credit contract has insufficient BNB to cover request

	> set return value > mark request as processed > increment markers

Issue_04	Edge-case during redelegation may allow malicious user to permanently brick withdrawals
Severity	**High**
Description	The redelegation flow will decrease the underlying BNB amount because a fee is taken on this procedure. If this is executed a few times, it is possible for the exchange rate to become negative (<1e18). If the exchange rate ever becomes negative, a user can simply request many withdrawals with 1 wei through different wallets. Due to the fact that the startBatchUndelegation is limited to a reasonable looping size, it is not possible to loop over thousands of iterations. If now the exchange rate is <1e18 by some means, this will result in 100wei of BNBx to eventually become zero BNB, thus resulting in zero shares as parameter for the undelegate function. This will result in a revert and will permanently DoS the withdrawal flow.
Recommendations	Consider setting a reasonable lower limit for withdrawal requests.
Comments / Resolution	**Failed resolution:** Redelegations will never work for any BNB amount < 5_000 BNB. While it is now ensured that the caller must provide a msg.value to counter any potential loss, the minimum delegation value of 1e18 was ignored: if (bnbAmount < minDelegationBNBChange) revert DelegationAmountTooSmall(); If we now consider the following check:

```
if (msg.value != getRedelegationFee(_amount)) revert
RedelegationFeeMismatch();

function getRedelegationFee(uint256 _amount) public view returns
(uint256) {

return (_amount * STAKE_HUB.redelegateFeeRate()) /
STAKE_HUB.REDELEGATE_FEE_RATE_BASE();

}
```

redelegateFeeRate = 2
REDELEGATE_FEE_RATE_BASE = 100_000

The redelegation fee for 100 BNB would be as follows:

100e18 * 2 / 100_000e18 = 2e15

It would therefore never work to redelegate BNB, only once the redelegationFee becomes 1e18, which is the case if BNB amount is 50_000.

Due to the fact that redelegations don't work anymore, this will increase the "Architectural issue can result in funds being locked in the contract" issue from low to high.

We recommend removing this check.

Resolution 3: The msg.value attachment of the function has been removed. This issue is considered as fixed.

Issue_05	Users may fail to claim because `completeBatchUndelegation` may result in less BNB than expected
Severity	**High**
Description	The `startBatchUndelegation` function calls `getSharesByPooledBNB` to convert the BNB amount into the operator's shares, and then undelegate the corresponding shares. The `getSharesByPooledBNB` function will round down, which may cause the `STAKE_HUB.undelegate` function to unlock less BNB than expected. Then, the `completeBatchUndelegation` function calls `STAKE_HUB.claim`, and the BNB obtained is less than `batchRequest.amountInBnb`. When the user claims, it will fail due to insufficient BNB. For example, suppose the operator's exchange rate is `1.2`. The `startBatchUndelegation` function expects to unlock `1e18` BNB, then the corresponding shares from `getSharesByPooledBNB` is `1e18 / 1.2 = 833333333333333333`. The `STAKE_HUB.undelegate` function unlocks `833333333333333333 * 1.2 = 999999999999999999` BNB. However `batchRequest.amountInBnb` is still `1e18`. When a user claims, it will fail due to insufficient BNB. This issue was already raised in the first iteration.
Recommendations	Since this issue is now also present in the second instance, we distance us from providing a code-based recommendation and rather recommend transferring some dust amount < 0.1 BNB directly to the contract (implement fallback) which then covers the worst-case discrepancy
Comments / Resolution	Resolved, the logic has been changed as follows: > calculate shares for desired BNB amount > calculate received BNB amount for shares

	> use the result from above and set it to amountInBnb

Issue_06	Architectural issue can result in funds being locked in the contract
Severity	**Medium**
Description	`StakeManagerV2` can delegate to multiple operators. Users will first delegate to the `preferredDepositOperator`. The admin controls the number of delegates for each operator by changing the `preferredDepositOperator`. The BNB assets held by BNBX are the sum of the delegates of all operators. If a user holds a large amount of BNBX, these BNBX correspond to the BNB of multiple operator delegates. When the user initiates a withdrawal, there may not be a single operator whose delegate quantity can meet the withdrawal requirements. This will cause the withdrawal to be unable to be processed and block the withdrawal queue. **Illustrated:** a) Alice deposits 100e18 BNB which is delegated to OP 1, Alice receives 100e18 BNBx b) Bob deposits 200e18 BNB which is delegated to OP1, Bob receives 200e18 BNBx c) PreferredDepositorOperator is changed to OP 2 d) Charles deposits 70e18 BNB which is delegated to OP2, Charles receives 70e18 BNBx e) Alice and Charles request a withdrawal, both withdrawals are batched into one batchWithdrawalRequest and taken from operator 1 f) After this request has been fulfilled, the following amounts are delegated: Operator 1 = 130e18; Operator 2 = 100e18 g) Bob requests a withdrawal for his 200e18 tokens

	h) The withdrawal process is now bricked because this request cannot be honored (the operator has insufficient delegated BNB). Fortunately, the owner can redelegate funds from Operator 2 to Operator 1 which then re-enables the processing.
Recommendations	Consider being very strict when working with different operators to ensure such discrepancies can never happen. There is no trivial code-sided solution for this problem.
Comments / Resolution	Acknowledged, however, due to the issue with redelegation, this issue will now be considered as **high instead of medium.** **Resolution 3: This has been fixed**

Issue_07	Malicious users can temporarily brick the queue by sybil'ing the maxActiveRequestsPerUsers
Severity	**Medium**
Description	In the previous audit we have recommended to implement a limit per user but also a lower threshold for amounts. The lower threshold was not implemented which means that users can simply create a script that seeds thousands of wallets and calls the requestWithdrawal function with 1 wei to artificially increase the length of the withdrawalRequests array. This means that governance needs to invoke the startBatchDelegation function until the queue is being emptied. The problem is that the queue can become very,very large, which then

	effectively breaks withdrawals for a long time.
Recommendations	Consider setting a reasonable lower limit of how much BNBx can be burned.
Comments / Resolution	Resolved.

Issue_08	Lack of ER update allows users to flash-theft tokens
Severity	**Medium**
Description	The `StakeManagerV2` contract uses `totalDelegated / totalSupply` as the exchange rate, where `totalDelegated` consists of two parts: the BNB staked by the user and the BNB earned from staking. The former is updated in real time every time BNBX is minted and burned, while the latter is updated by externally calling the `updateER` function. If the `updateER` function is not called externally, the exchange rate will lag behind. Users may use this to steal the staking income. For example, suppose that the current `totalDelegated` and `totalSupply` are both `1e18`. 1. After a period of time, the total value of the stake increases to `1.2e18` BNB, but `updateER` is not called during this period 2. UserA stakes `1e18` BNB and gets `1e18` BNBX, `totalDelegated` and `totalSupply` become `2e18` 3. UserA calls `updateER` to synchronize the staking income, and `totalDelegated` increases to `2.2e18` 4. UserA's `1e18` BNBX is now worth `1.1e18` BNB

	Similarly, this would result in users receiving less funds than they should if the ER is not updated before the startBatchUndelegation function.
Recommendations	Consider updating the ER at the beginning of these functions.
Comments / Resolution	Acknowledged: During delegate, the ER is not updated, which means users can simply deposit, update the ER and then receive more tokens back than they initially provided. Depending on the time since the last update, this can result in a huge gain for an exploiter. The ER should be updated at the beginning of the delegate function as well. *Stader comment: As there is a 7-day withdrawal delay, the attacker's funds would also be locked without generating any yield, making this attack not beneficial to the attacker. Hence, we believe this shouldn't be an issue. Additionally, it could significantly increase the gas barrier for the delegate, as updateER is expensive.*

Issue_09	setFeeBps function will alter fee in hindsight
Severity	**Medium**
Description	The setFeeBps function allows governance to change the fee which is taken on rewards. Due to the fact that the ER is not updated beforehand, such an update may alter the fee in hindsight. If for example the ER was not updated for 1 week and the fee was 5%, the fee can now be updated to 50%. However, this 50% will now apply on the whole past week.
Recommendations	Consider updating the ER before the fee is updated.

Comments / Resolution	Resolved.

Issue_10	Users may lose anticipated funds if redelegation happens after a withdrawal has been requested
Severity	**Low**
Description	Whenever users request a withdrawal, there is absolutely no way to cancel this withdrawal again. If a redelegation of funds happens after a withdrawal has been requested this will decrease the underlying BNB amount and results in these requests receiving less BNB as initially expected.
Recommendations	Consider communicating such a redelegation one week before with the community.
Comments / Resolution	Resolved, this issue has been inherently resolved due to the fact that a msg.value must be provided with the redelegation. However, the issue which was introduced within the redelegation flow must be fixed.

Issue_11	Dust may remain within operators
Severity	**Low**
Description	As already explained, the withdrawal amount from operators will round down whenever the share value is determined. This may result in leftover amounts being stuck in the operator.
Recommendations	Since this issue was also present in the previous iteration, we do not recommend any further code change. If we incorporate the new logic into the context, the last user may just not be able to redeem a few wei of shares.

Comments / Resolution	Acknowledged.

Issue_12	Incorrect NATSPEC about access control for completeBatchUndelegation
Severity	**Low**
Description	The NATSPEC for this function indicates that it should solely be callable by the operator: /// @dev This function can only be called by an address with the OPERATOR_ROLE. This is however not the case.
Recommendations	Consider removing this comment or implementing an access control mechanism.
Comments / Resolution	Resolved.

Issue_13	The boosting rewards may be arbitraged
Severity	**Informational**
Description	The `delegateWithoutMinting` function has two uses: 1. Migrate assets from `StakeManager` to `StakeManagerV2` 2. Admins donate BNB to `StakeManagerV2` to increase rewards This function, when used to increase rewards, can be used by attackers for arbitrage because it will immediately increase the

	exchange rate. Once the attacker finds the `delegateWithoutMinting` tx in the memory pool, he can mint BNBX in advance. After the `delegateWithoutMinting` tx is executed, the value of his BNBX will immediately increase.
Recommendations	Consider either accepting this risk and staying reasonable with one-time reward boosting or implement a mechanism which linearly vests these rewards. The latter scenario needs additional validation.
Comments / Resolution	Acknowledged.

Issue_14	Treasury fee will be slightly larger after share minting
Severity	**Informational**
Description	In the scenario where the treasury fee is not 100%, the mathematical calculation will result in the fee becoming slightly larger due to the fact that the pooled bnb value is used for the arithmetic operation. **Illustrated:** BNBx.supply = 100e18 totalDelegated = 100e18 underlyingBNB = 110e18 feeBps = 5000 Therefore, there is currently an unupdated profit of 10e18. a) Calculate the feeInBnb: -> (totalPooledBnb - totalDelegated) * feeBps / 10000 -> (110e18 - 100e18) * 5000 / 10000 -> 5e18 b) Convert this bnb amount to the corresponding share value:

	-> 5e18 * 100e18 / 100e18 -> 5e18 c) After the update, calculate the value of these 5 minted shares: -> 5e18 * 110e18 / 105e18 -> 5.23e18 The treasury effectively received 0.23e18 more shares than expected.
Recommendations	There are three points to consider: a) This action is in favor of the protocol b) The math works correct if the fee is 100% c) This is likely a design choice Therefore, we are the decision that the math should not be adjusted and this issue can be safely acknowledged.
Comments / Resolution	Acknowledged.

StakeManager

Bailsec was tasked with a trivial check for the migrateFunds function in the StakeManager. The function in question can be found in the following diffcheck:
https://www.diffchecker.com/O2JRmIWx/

https://github.com/stader-labs/bnbX/blob/2bba18c23cf0b2fdf03e80be7cd0976479bb3d1d/contracts/StakeManager.sol#L336

This function simply allows any address with the DEFAULT_ADMIN_ROLE to withdraw BNB in the size of “depositsInContract”. This value trivially represents how much BNB is sitting in the contract and was not yet delegated.
Delegated funds cannot be transferred out by this function.

Additionally it needs to be mentioned that this function can be called multiple times. Therefore we just recommend adding a parameter which simply allows the caller to specify how much funds should be exactly withdrawn. This will increase flexibility.