

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

Introduction

The SteakBank.Finance is a staking derivatives DeFi platform which enables users to participate in Binance Smart Chain staking on BSC and unlock the staked BNB liquidity by offering users LBNB. The DeFi platform also provides liquidity farming to encourage users to lock their SBF and offer liquidity in decentralized exchange. As for the governance mechanism, it is very similar to most existing DeFi platforms.

Contract Code:

<https://github.com/steakbankfinance/steakbank-contract/tree/master/contracts>

Covered contracts:

UnstakeVault.sol
SteakBankImpl.sol
StakingRewardVault.sol
SBF.sol
LBNB.sol
CommunityTaxVault.sol
Ownable.sol
SBFRewardVault.sol
BEP20.sol
FarmRewardLock.sol
FarmingCenter.sol
BlindFarmingCenter.sol

Commit hash: 330dd5924e2b2ac6218fae96afc2cc3e4885e59b

Summary of Findings

All code is implemented by solidity. The implementation is not very complex. 360 Safeguard discovered a number of vulnerabilities ranging in severity from informational to high. The issues are listed in this report. We recommend addressing all the listed findings before deployment.

ID	Description	Severity	Status
1	Denial of service in batch claim If a contract(without implementing receive method) calls stake	High	Fixed
2	Possible reentrancy attack	Medium	Fixed
3	Accelerating unstake mature burns unexpected too many SBF	Medium	Fixed

4	Unstake tiny amount to wastes gas fee of volunteers who will call batchClaim	Medium	Fixed
5	Using send and transfer for native token transfers	Low	Acknowledged
6	Call an external function before internal work done	Low	Acknowledged
7	Missing input parameter validation in FarmingCenter	Low	Fixed
8	Magnified precision truncation	Low	Acknowledged
9	Symbolic constant names are not uppercase	Informational	Acknowledged
10	Use try catch in swap	Informational	Acknowledged
11	Not using constructors	Informational	Acknowledged
12	Redundant spaces	Informational	Fixed
13	Messy variable modifier order	Informational	Fixed
14	The init() functions can be declared external instead of public	Informational	Acknowledged
15	Iteration might cost unexpected amount of gas	Informational	Acknowledged
16	Missing comment for function parameters	Informational	Acknowledged
17	Error messages in required sentences are in different styles.	Informational	Acknowledged

360 Safeguard

Findings

1. Denial of service in batch claim

Type	Severity	Location
Business Model	High	SteakBankImpl: batchClaimPendingUnstake

Description

In [SteakBankImpl.sol, L337](#), there is a loop and the `claim` method of `unstakeVault` is called in the loop. In the `claim` method of `unstakeVault` ([UnstakeVault.sol, L26](#)), the native token will be transferred to the staker. If the staker is a contract (without implementing `receive` method), this method will always fail to execute. Therefore, if there is a contract (without implementing `receive` method) calls the `unstake` method ([SteakBankImpl.sol, L198](#)) will cause the Denial of Service in `batch claim`. Besides, the contract can implement a `receive` method which will cost huge gas which might be larger than `block gasLimit`, thus this is a Denial of Service attack too.

Recommendation

Reject all contracts to call `unstake` method.

2. Possible reentrancy attack

Type	Severity	Location
Logic Issue	Medium	CommunityTaxVault: claimBNB

Description

In [CommunityTaxVault.sol, L56](#), the `claimBNB(...)` method, does not check the reentrant calls. There may be a risk of reentrancy attack.

Recommendation

Add a `nonReentrant` modifier to the `claimBNB()` function.

3. Accelerating unstake mature burns unexpected too many SBF

Type	Severity	Location
Business Model	Medium	SteakBankImpl: accelerateUnstakedMature

--	--	--

Description

In [SteakBankImpl.sol](#), L264, the `accelerateUnstakedMature(...)` method, accumulates `sbfBurnAmount` as `sbfBurnAmount = sbfBurnAmount.add(priorUnstake.amount.mul(priceToAccelerateUnstake))`, because `priceToAccelerateUnstake` is defined as `uint256`, `sbfBurnAmount > priorUnstake.Amount`, it is expensive if the `SBF` price rises.

Recommendation

Add `PRECISION` on calculating `sbfBurnAmount`.

4. Low range for required unstake amount

Type	Severity	Location
Business Model	Low	SteakBankImpl: unstake

Description

In [SteakBankImpl.sol](#), L225, the `unstake(...)` method does not set a minimum unstake amount. Attackers may initiate many small unstake transactions causing normal users to spend more when calling the `accelerateUnstakedMature(...)` method.

Recommendation

Set minimum unstake amount.

5. Using transfer for native token transfers

Type	Severity	Location
Logic Issue	Low	UnstakeVault: L30 StakingRewardVault: L30 CommunityTaxVault: L30

Description

In [UnstakeVault: L30](#), [StakingRewardVault: L30](#), [CommunityTaxVault: L30](#), the `transfer` is used to transfer BNB. This is not safe if the recipient is a contract address. If the contract doesn't implement the `receive` method, then the transfer will be reverted.

Recommendation

Use contract calls with value instead of transfer.

6. Call an external function before internal work done

Type	Severity	Location
Logic Issue	Low	FarmingCenter.sol: L258

Description

In [FarmingCenter.sol: L258](#), the emergencyWithdraw call [safeTransfer](#) before setting user.amount and user.rewardDebt to zero. We recommend completing the internal work first and calling the external function at last.

Recommendation

Move L260 and L261 before calling [safeTransfer](#).

7. Missing input parameter validation

Type	Severity	Location
Logic Issue	Low	FarmingCenter: updatePool, deposit, withdraw, emergencyWithdraw

Description

In [FarmingCenter.sol](#), the [updatePool](#), [deposit](#), [withdraw](#), [emergencyWithdraw](#) method does not validate parameter: `_pid` whether less than the length of `poolInfo`, `_pid` greater than `poolInfo.length` may cause unexpected results.

Recommendation

Add validation of the assumed input lengths.

8. Magnified precision truncation

Type	Severity	Location
Business Model	low	SteakBankImpl: estimateSBFCostForAccelerate, accelerateUnstakedMature

Description

In [SteakBankImpl.sol](#), L250: `estimateSBFCostForAccelerate(...)` and L264: `accelerateUnstakedMature(...)`, `sbfburnAmount` is accumulated by the consumed amount in each step, and `step_amount` is truncated by `PRICE_TO_ACCELERATE_UNSTAKE_PRECISION`. Therefore, The deviation of `sbfburnAmount` will be amplified, especially when `PRICE_TO_ACCELERATE_UNSTAKE_PRECISION` is great enough.

Recommendation

Don't divide `PRICE_TO_ACCELERATE_UNSTAKE_PRECISION` during the sum calculation and divide `PRICE_TO_ACCELERATE_UNSTAKE_PRECISION` after the sum calculation.

9. Symbolic constant names are not uppercase

Type	Severity	Location
Bestpractice	Informational	SteakBankImpl : L23, L24

Description

In [SteakBankImpl.sol](#), L23 and L24, There are two constants named `mininum_stake_amount` and `mininum_unstake_amount`. But symbolic constant names should commonly be written in uppercase so they can be readily distinguished from lower case variable names.

Recommendation

1. Rename `mininum_stake_amount` to `MINIMUM_STAKE_AMOUNT`.
2. Rename `mininum_unstake_amount` to `MINIMUM_UNSTAKE_AMOUNT`.

10. Use try catch in swap

Type	Severity	Location
Bestpractice	Informational	CommunityTaxVault : L83 and L94

Description

In [CommunityTaxVault](#): L83 and L94, `buyAndBurnSBF` method will call an external contract to swap BNB for SBF and swap LBNB for SBF. The external contract call might be failed, it would be better if the code can elegantly handle the uncentains failure.

Recommendation

Use try catch to to handle the uncertain external function call.

11. Not using constructors

Type	Severity	Location
Bestpractice	Informational	BlindFarmingCenter.sol: L48 FarmingCenter.sol: L47 FarmRewardLock.sol: L41

Description

[BlindFarmingCenter: L48](#) uses a public `initialize` method instead of a constructor to set initial state. Anyone can call the `initialize` method to set initial state. The contract deployer needs to call the `initialize` method as soon as possible.

Recommendation

Use constructor instead of `initialize` method to set initial state

14. The `initialize()` functions can be declared external instead of public and use initializer modifier in `openzappelin`

Type	Severity	Location
Bestpractice	Informational	BlindFarmingCenter.sol: L48 FarmingCenter.sol: L47 FarmRewardLock.sol: L41

Description

The `initialize` method in [BlindFarmingCenter.sol: L48](#), [FarmingCenter.sol: L47](#), [FarmRewardLock.sol: L41](#) will only be called by other contracts or external accounts, and there is no internal method to call it, so external is more suitable than public because it can save gas and explicit indicate who will call it.

The `initialize` method can only be called for once. `Openzappelin` has implemented a mature initializer modifier which can perfectly ensure this. However, the `initialize` method implements an `initialized` flag which is not elegant.

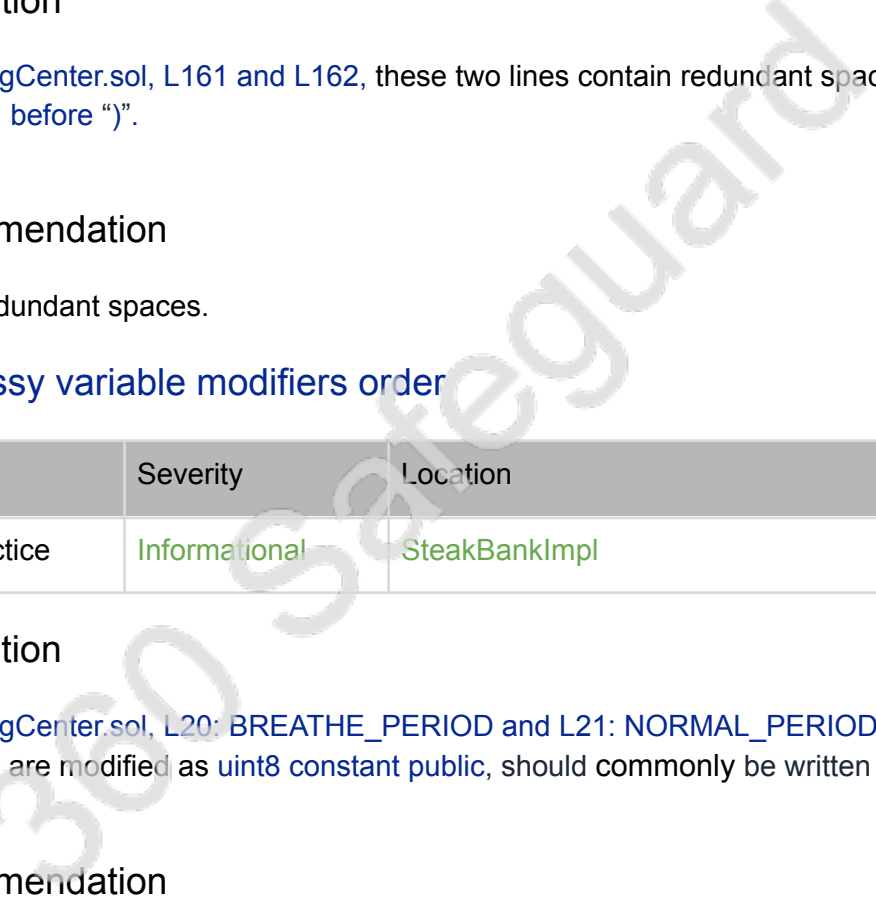
Recommendation

Use “external” to replace “public”, and use the Openzappelin [initializer](#) modifier instead of the [initialized](#) flag.

12. Redundant spaces

Type	Severity	Location
Bestpractice	Informational	FarmingCenter: getMultiplier

Description

In [FarmingCenter.sol](#), [L161](#) and [L162](#), these two lines contain redundant spaces. [L161](#), after “>=”; [L62](#), before “)”.


Recommendation

Delete redundant spaces.

13. Messy variable modifiers order

Type	Severity	Location
Bestpractice	Informational	SteakBankImpl

Description

In [FarmingCenter.sol](#), [L20: BREATHE_PERIOD](#) and [L21: NORMAL_PERIOD](#), these two constants are modified as [uint8 constant public](#), should commonly be written as [uint8 public constant](#).

Recommendation

Adjust the order of modifiers.

15. Iteration might cost unexpected amount of gas

Type	Severity	Location
Bestpractice	Informational	SteakBankImpl: L342

Description

[batchClaimPendingUnstake](#) will be out of gas if the batchSize is too large. In solidity, it is very dangerous to allow large loop iteration. It is recommended to let each individual staker to claim their own unstaker. Thus there will be no out of gas issue.

Recommendation

Remove [batchClaimPendingUnstake](#) and add [claimPendingUnstake](#) method, [claimPendingUnstake](#) will only try to fulfill only one unstake and there will be no out gas concern.

16. Missing comment for function parameters

Type	Severity	Location
Bestpractice	Informational	SteakBankImpl.sol BlindFarmingCenter.sol FarmingCenter.sol FarmRewardLock.sol

Description

It is recommended to add comments for parameters like [BEP20.sol L103](#). The above four contracts are the core contracts of this project, however, there is no any parameters comment. The parameter comments can help users to understand the detailed mechanism and facilitate communities to build useful tools to help the community development.

Recommendation

Add parameter comments for the four contracts.

17. Error messages in required sentences are in different styles.

Type	Severity	Location
Bestpractice	Informational	SteakBankImpl : L238 , L275 , L276 , L279 , L285 , L297 , L311 , L313 , L365 , L375 , L377 , L394 , L395 , L396

Description

It would be better to assemble the required error message in this format: method name + detailed failure reason, such as [SteakBankImpl](#): [L164](#), [L172](#). However, the other required

error messages in [SteakBankImpl](#) don't follow the format. The error message format can explicitly tell users the failure position and failure reason.

Recommendation

Refactor all require error messages to follow the recommended format.

360 Safeguard

Test Results

Contract: BlindFarmingCenter Contract

- ✓ Test Deposit LBNB to BlindFarmingCenter (3610ms)
- ✓ Test Deposit LBNB to BlindFarmingCenter (3820ms)

Contract: SteakBank Contract

- ✓ Test Stake (1444ms)
- ✓ Test Unstake (1193ms)
- ✓ Test rebaseLBNBToBNB (3380ms)
- ✓ Test resendBNBToBCStakingTSS (552ms)
- ✓ Test transfer admin (598ms)
- ✓ Test pause (783ms)

Contract: Governance Contract

- ✓ Test Init Governor Contract (13079ms)
- ✓ Test Submit, Vote, Queue and Execute Proposal (14052ms)
- ✓ Test Cancel Proposal (12448ms)

Contract: CommunityTaxVault Contract

- ✓ Test Claim BNB (472ms)
- ✓ Test Transfer governorship (401ms)
- ✓ Test buyAndBurnSBF (1709ms)
- ✓ Test change LBNBAddr, SBFAddr and PancakeRouterAddr (802ms)

Contract: FarmingCenter Contract

- ✓ Test Deposit SBF (3563ms)
- ✓ Test Deposit LBNB (3237ms)
- ✓ Test Reward Lock (3045ms)
- ✓ Test Farming End (6870ms)

19 passing (1m)

Code Coverage

The total test coverage for the Solidity smart contracts was 84.03% which is very pleasant. However, some contracts coverage is relatively low, for example SBF Contract. We recommend that the coverage of all contracts should be more than 80%. We have checked all unit tests, and we suggest adding more test cases to governance because the current test cases can't simulate complex user behavior.

contracts/farm/BlindFarmingCenter.sol FarmRewardLock 86.05%
contracts/farm/FarmRewardLock.sol FarmRewardLock 95.56%
contracts/farm/FarmingCenter.sol FarmingCenter 95.56%
contracts/UnstakeVault.sol UnstakeVault 85.71%
contracts/CommunityTaxVault.sol CommunityTaxVault 100%
contracts/StakingRewardVault.sol StakingRewardVault 85.71%
contracts/SteakBankImpl.sol SteakBankImpl 87.58%
contracts/LBNB.sol LBNB 100%
contracts/SBF.sol SBF 69.09%
contracts/governance/Governor.sol Governor 78.22%
contracts/governance/Timelock.sol Timelock 85.37%
contracts/lib/Ownable.sol Ownable 33.33%
contracts/lib/BEP20.sol BEP20 85.71%

Summary of the audit

Overall the code is well implemented and clear on what it's supposed to do for each function. The mechanism about stake, unstake, as well as farming is quite clear and they shouldn't bring major issues. My final recommendation would be adding more comments and documents about method parameters and the detailed stake and unstake mechanism. This is a secure contract that will store the funds safely while it's working.

360 Safeguard

Email

wangzheng1@360.cn

360 Safeguard