BEOSIN
Blockchain Security

Smart contract security audit report

Audit Number：202104281601

Smart Contract Name：

SteakBankImpl

Smart Contract Link：

https://github.com/steakbankfinance/steakbank-contract/blob/master/contracts/SteakBankImpl.sol

Start Commit：

f66249215ca36c9af961c5933ce8ed9fddb8c936

End Commit：

254e58fedc90afbae421dc5e2ac1bf3ac1d7c075

Start Date：2021.04.26

Completion Date：2021.04.28

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass

		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract SteakBankImpl, including Coding Standards, Security, and Business Logic. **The SteakBankImpl contract passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.

- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

3.1 Business analysis of Contract SteakBankImpl

(1) *stake* function

- Description: The *stake* method implements the function of users staking BNB assets, and users can initiate calling this function to send BNB to the contract to obtain IBNB. The actual amount of the user will remove the fraction. The BNB sent by the user will be sent to the address of the project party on another chain through a cross-chain contract. The investment management is carried out by the project party.

```
202     function stake(uint256 amount) nonReentrant mustInPeriod(NORMAL_PERIOD) notContract whenNotPaused external payable returns (bool) {
203
204         uint256 miniRelayFee = ITokenHub(TOKENHUB_ADDR).getMiniRelayFee();
205
206         require(msg.value == amount.add(miniRelayFee), "msg.value must equal to amount + miniRelayFee");
207         require(amount%1e10==0 && amount>=MINIMUM_STAKE_AMOUNT, "stake amount must be N * 1e10 and more than 1:BNB");
208
209         uint256 stakeFee = amount.mul(stakeFeeMolecular).div(stakeFeeDenominator);
210         communityTaxVault.transfer(stakeFee);
211         uint256 stakeAmount = amount.sub(stakeFee);
212         lbnbMarketCapacityCountByBNB = lbnbMarketCapacityCountByBNB.add(stakeAmount);
213         uint256 lbnbAmount = stakeAmount.mul(EXCHANGE_RATE_PRECISION).div(lbnbToBNBExchangeRate);
214
215         uint256 stakeAmountDust = stakeAmount.mod(1e10);
216         if (stakeAmountDust != 0) {
217             unstakeVault.transfer(stakeAmountDust);
218             stakeAmount = stakeAmount.sub(stakeAmountDust);
219         }
220
221         ITokenHub(TOKENHUB_ADDR).transferOut{value:miniRelayFee.add(stakeAmount)}(ZERO_ADDR, bcStakingTSS, stakeAmount, uint64(block.timestamp + 3600));
222
223         IMintBurnToken(LBNB).mintTo(msg.sender, lbnbAmount);
224         emit LogStake(msg.sender, lbnbAmount, stakeAmount);
225
226         return true;
227     }
```

Figure 1 source code of stake

- Safe suggestion: User assets are stored in an address, if the private key is lost, it will result in asset loss.
- Fixed result: The project party will use the TSS management address to ensure the security of the private key.
- Result: Pass

(2) *unstake* function

- Description: The *unstake* method implements the function of user settlement and profit. The user can call the *unstake* function to destroy the IBNB to obtain an Unstake lock-up information, which can be used to exchange BNB. The specific exchange ratio is calculated according to the exchange ratio when unstake is called.

```
229     function unstake(uint256 amount) nonReentrant mustInPeriod(NORMAL_PERIOD) notContract whenNotPaused external returns (bool) {
230         require(amount>=MINIMUM_UNSTAKE_AMOUNT, "unstake amount must be more than 0.8:LBNB");
231         uint256 unstakeFee = amount.mul(unstakeFeeMolecular).div(unstakeFeeDenominator);
232         IERC20(LBNB).safeTransferFrom(msg.sender, communityTaxVault, unstakeFee);
233
234         uint256 unstakeAmount = amount.sub(unstakeFee);
235         IERC20(LBNB).safeTransferFrom(msg.sender, address(this), unstakeAmount);
236         IMintBurnToken(LBNB).burn(unstakeAmount);
237
238         uint256 bnbAmount = unstakeAmount.mul(lbnbToBNBExchangeRate).div(EXCHANGE_RATE_PRECISION);
239         bnbAmount = bnbAmount.sub(bnbAmount.mod(1e10));
240         lbnbMarketCapacityCountByBNB = lbnbMarketCapacityCountByBNB.sub(bnbAmount);
241         unstakesMap[tailIdx] = Unstake({
242             staker: msg.sender,
243             amount: bnbAmount,
244             timestamp: block.timestamp
245         });
246         uint256[] storage unstakes = accountUnstakeSeqsMap[msg.sender];
247         unstakes.push(tailIdx);
248
249         emit LogUnstake(msg.sender, unstakeAmount, bnbAmount, tailIdx);
250         tailIdx++;
251         return true;
252     }
```

Figure 2 source code of *unstake*

- Result: Pass

(3) *accelerateUnstakedMature* function

- Description: The *accelerateUnstakedMature* method implements the function of users to unlock Unstake in advance. The user can call this method to unlock and jump in the queue by destroying SBF tokens. But only supports jump in the queue on the same day.

```
275     function accelerateUnstakedMature(uint256 unstakeIndex, uint256 steps, uint256 sbfMaxCost) nonReentrant whenNotPaused external returns (bool) {
276         require(steps > 0, "accelerate steps must be greater than zero");
277         require(unstakeIndex.sub(steps)>=headerIdx && unstakeIndex<tailIdx, "unstakeIndex is out of valid accelerate range");
278 
279         Unstake memory unstake = unstakesMap[unstakeIndex];
280         require(unstake.staker==msg.sender, "only staker can accelerate itself");
281         uint256 timestampThreshold = unstake.timestamp.sub(unstake.timestamp.mod(86400));
282 
283         uint256 sbfBurnAmount = unstake.amount.mul(steps).mul(priceToAccelerateUnstake);
284         for (uint256 idx = unstakeIndex.sub(1) ; idx >= unstakeIndex.sub(steps); idx--) {
285             Unstake memory priorUnstake = unstakesMap[idx];
286             require(priorUnstake.timestamp>=timestampThreshold, "forbid to exceed unstake in prior day");
287             unstakesMap[idx+1] = priorUnstake;
288             sbfBurnAmount = sbfBurnAmount.add(priorUnstake.amount.mul(priceToAccelerateUnstake));
289             uint256[] storage priorUnstakeSeqs = accountUnstakeSeqsMap[priorUnstake.staker];
290             bool found = false;
291             for(uint256 i=0; i < priorUnstakeSeqs.length; i++) {
292                 if (priorUnstakeSeqs[i]==idx) {
293                     priorUnstakeSeqs[i]=idx+1;
294                     found = true;
295                     break;
296                 }
297             }
298             require(found, "failed to find matched unstake sequence");
299         }
300         sbfBurnAmount = sbfBurnAmount.div(PRICE_TO_ACCELERATE_UNSTAKE_PRECISION);
301 
302         uint256[] storage unstakeSeqs = accountUnstakeSeqsMap[msg.sender];
303         unstakesMap[unstakeIndex.sub(steps)] = unstake;
304         bool found = false;
305         for(uint256 idx=0; idx < unstakeSeqs.length; idx++) {
306             if (unstakeSeqs[idx]==unstakeIndex) {
307                 unstakeSeqs[idx] = unstakeIndex.sub(steps);
308                 found = true;
309                 break;
310             }
311         }
312         require(found, "failed to find matched unstake sequence");
313 
314         require(sbfBurnAmount<=sbfMaxCost, "cost too much SBF");
315         IERC20(SBF).safeTransferFrom(msg.sender, address(this), sbfBurnAmount);
316         IMintBurnToken(SBF).burn(sbfBurnAmount);
317 
318         emit AcceleratedUnstakedBNB(msg.sender, unstakeIndex);
319 
320         return true;
```

Figure 3 source code of *accelerateUnstakedMature*

- Result: Pass

(4) *batchClaimPendingUnstake* functions

- Description: The *batchClaimPendingUnstake* method implements the function of users unlocking Unstake and obtaining BNB. The user can unlock a specified number of Unstakes and obtain the recorded BNB by calling this function.

```
343     function batchClaimPendingUnstake(uint256 batchSize) nonReentrant whenNotPaused external {
344         for(uint256 idx=0; idx < batchSize && headerIdx < tailIdx; idx++) {
345             Unstake memory unstake = unstakesMap[headerIdx];
346             uint256 unstakeBNBAmount = unstake.amount;
347             if (unstakeVault.balance < unstakeBNBAmount) {
348                 return;
349             }
350             delete unstakesMap[headerIdx];
351             uint256 actualAmount = IVault(unstakeVault).claimBNB(unstakeBNBAmount, unstake.staker);
352             require(actualAmount==unstakeBNBAmount, "amount mismatch");
353             emit ClaimedUnstake(unstake.staker, unstake.amount, headerIdx);
354 
355             uint256[] storage unstakeSeqs = accountUnstakeSeqsMap[unstake.staker];
356             uint256 lastSeq = unstakeSeqs[unstakeSeqs.length-1];
357             if (lastSeq != headerIdx) {
358                 bool found = false;
359                 for(uint256 index=0; index < unstakeSeqs.length; index++) {
360                     if (unstakeSeqs[index]==headerIdx) {
361                         unstakeSeqs[index] = lastSeq;
362                         found = true;
363                         break;
364                     }
365                 }
366                 require(found, "failed to find matched unstake sequence");
367             }
368             unstakeSeqs.pop();
369 
370             headerIdx++;
371         }
372     }
```

Figure 4 source code of *batchClaimPendingUnstake*

- Result: Pass

(5) *rebaseLBNBToBNB* functions

- Description: The *rebaseLBNBToBNB* method implements the function of adjusting the relative price of IBNB and BNB. When this function is called, the BNB balance in the stakingRewardVault contract will be added to the unstakeVault contract, and the newly added BNB will be equally distributed to IBNB. Therefore, as long as the stakingRewardVault contract continue to receive BNB, and the price of IBNB relative to BNB will continue to rise.

```
374     function rebaseLBNBToBNB() whenNotPaused external returns(bool) {
375         uint256 rewardVaultBalance = stakingRewardVault.balance;
376         require(rewardVaultBalance>0, "stakingRewardVault has no BNB");
377         uint256 actualAmount = IVault(stakingRewardVault).claimBNB(rewardVaultBalance, unstakeVault);
378         require(rewardVaultBalance==actualAmount, "reward amount mismatch");
379 
380         uint256 lbnbTotalSupply = IERC20(LBNB).totalSupply();
381         lbnbMarketCapacityCountByBNB = lbnbMarketCapacityCountByBNB.add(rewardVaultBalance);
382         if (lbnbTotalSupply == 0) {
383             lbnbToBNBExchangeRate = EXCHANGE_RATE_PRECISION;
384         } else {
385             lbnbToBNBExchangeRate = lbnbMarketCapacityCountByBNB.mul(EXCHANGE_RATE_PRECISION).div(lbnbTotalSupply);
386         }
387         emit LogUpdateLBNBToBNBExchangeRate(lbnbTotalSupply, lbnbMarketCapacityCountByBNB, lbnbToBNBExchangeRate);
388         return true;
389     }
```

Figure 5 source code of *rebaseLBNBToBNB*

- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contract SteakBankImpl. The problems found by the audit team during the audit process have been notified to the project party and reached an agreement on the repair results.The contract SteakBankImpl passed all audit items, The overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com