HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Bitcoin SB
Date: October 11th, 2021

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Bitcoin SB.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Staking
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Deployed contract	1. https://etherscan.io/address/0xfac49aa6f7c7184f43a56034b4ddf5db240306e8#code 2. https://etherscan.io/address/0x78e3dd527d0f9d2c64e5ac2f7c1ec9384da58803#code 3. https://etherscan.io/address/0x7e9ff3e4e081f0af9186145de486d113f96977a1#code
Technical Documentation	NO
JS tests	NO
Timeline	09 OCTOBER 2021 - 11 OCTOBER 2021
Changelog	11 OCTOBER 2021 - INITIAL AUDIT

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Bitcoin SB (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contract and its code review conducted between October 9th, 2021 - October 11th, 2021.

Scope

The scope of the project is smart contracts in the repository:

Deployed Code:
https://etherscan.io/address/0xfac49aa6f7c7184f43a56034b4ddf5db240306e8#code
https://etherscan.io/address/0x78e3dd527d0f9d2c64e5ac2f7c1ec9384da58803#code
https://etherscan.io/address/0x7e9ff3e4e081f0af9186145de486d113f96977a1#code
Technical Documentation: No
JS tests: No
Contracts:
BSB_StakingStrongStakers
BSB_StakingSoftStakers
BSB_StakingSmartStakers

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

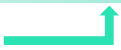
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Executive Summary

According to the assessment, the Customer's smart contracts are secured.

Insecure	Poor secured	Secured	Well-secured

You are here



Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

As a result of the audit, security engineers found **1** high and **4** low severity issues.

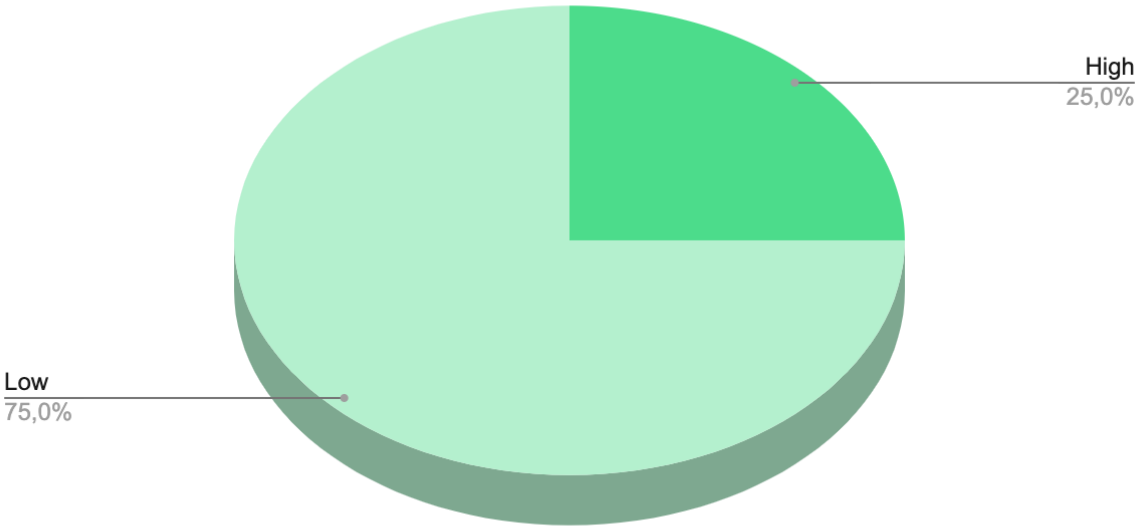
Notice:

A lot of loops could affect the functions "End" (high gas fee) and "getTotalRewards (unavailability) as soon as many users will do their stakes.

Customer acknowledged of the possible functions dysfunction (End [high gas fee], getTotalRewards [unavailibility]) because of using cycles over the stakeholders array.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Graph 1. The distribution of vulnerabilities after the audit.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Audit overview

Critical

No critical issues were found.

High

Possible out-of-order function

Doing a loop through all the stakeholders and processing them could burn a lot of gas. While having that on the writable only Admin functions may be risky in the meaning of the gas amount but on the view-only functions, it will make those functions inaccessible.

Recommendation: Please do not use looping through the list of holders, try to rely on maths instead. Take a look at SushiSwap's MasterChef for reference.

Lines: #393-395
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
for(uint i = 0; i < holders.length(); i = i.add(1)){
    _aux = holders.at(i);
    rewardEnded[_aux] = getPendingRewards(_aux);
```

Lines: #415-417
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
for(uint i = 0; i < holders.length(); i = i.add(1)){
    totalPending = totalPending.add(getPendingRewards(holders.at(i)));
}
```

Status: Customer uknowledged of the possible functions out-of-order because of using cycles over the stakeholders array.

Medium

No medium issues were found.

Low

1. **Lines:** BSB_StakingStrongStakers#336-345

```
/*
 * @dev Pool size
 */
uint public constant maxPoolSize = 500000000000000000000000;
uint public availablePoolSize = 500000000000000000000000;

/*
 * @dev Total rewards
 */
uint public constant totalRewards = 400000000000000000000000;
```

Lines: BSB_StakingSoftStakers#336-345

```
/*
 * @dev Pool size
 */
uint public constant maxPoolSize = 10000000000000000000000;
uint public availablePoolSize = 10000000000000000000000;

/*
 * @dev Total rewards
 */
uint public constant totalRewards = 50000000000000000000000;
```

Lines: BSB_StakingStrongStakers#336-345

```
/*
 * @dev Pool size
 */
uint public constant maxPoolSize = 50000000000000000000000;
uint public availablePoolSize = 50000000000000000000000;

/*
 * @dev Total rewards
 */
uint public constant totalRewards = 200000000000000000000000;
```

2. Using SafeMath for Solidity >= 0.8.0

   Starting the solc version 0.8.0, solidity already has a built-in math over and underflow validation. Using additional validation just spends gas.

   Recommendation: don't use SafeMath with asserts on Solidity >= 0.8.0

Lines: #5-14
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
pragma solidity >=0.8.0;

// SPDX-License-Identifier: BSD-3-Clause

library SafeMath {
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }
```

3. State variables that could be declared constant

   Constant state variables should be declared constant to save gas.

   Recommendation: Add the **constant** attributes to state variables that never change.

Lines: BSB_StakingStrongStakers#368-369

```
uint public number_intervals = 6;
uint public duration_interval = 30 days;
```

Lines: BSB_StakingStrongStakers#313

```
uint public rewardRate = 104000;
```

Lines: BSB_StakingStrongStakers#374

```
uint extraAPY = 10400; // 2% extra weekly
```

Lines: BSB_StakingStrongStakers#379

```
uint percent_claim = 4; // 20% of weekly rewards earned
```

Lines: BSB_StakingSoftStakers#368-369

```
uint public number_intervals = 3;
uint public duration_interval = 20 days;
```

Lines: BSB_StakingSoftStakers#313

```
uint public rewardRate = 26000;
```

Lines: BSB_StakingSoftStakers#374

```
uint extraAPY = 10400; // 2% extra weekly
```

Lines: BSB_StakingSoftStakers#379

```
uint percent_claim = 1; // 20% of weekly rewards earned
```

Lines: BSB_StakingSmartStakers#368-369

```
uint public number_intervals = 4;
uint public duration_interval = 30 days;
```

Lines: BSB_StakingSmartStakers#313

```
uint public rewardRate = 52000;
```

Lines: BSB_StakingSmartStakers#374

```
uint extraAPY = 10400; // 2% extra weekly
```

Lines: BSB_StakingSmartStakers#379

```
uint percent_claim = 2; // 20% of weekly rewards earned
```

4. A public function that could be declared external

   public functions that are never called by the contract should be declared **external** to save gas.

   Recommendation: Use the **external** attribute for functions never called from the contract.

Lines:#389
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function end() public onlyOwner returns (bool){
```

Lines:#562
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function getNumberOfHolders() public view returns (uint) {
```

Lines:#566
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function deposit(uint amountToStake) public returns (bool){
```

Lines:#657
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function withdraw2(uint amountToWithdraw) public returns (bool){
```

Lines:#686
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function withdraw(uint amountToWithdraw) public returns (bool){
```

Lines:#713
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function getTimeToWithdraw(address _staker) public view returns (uint){
```

Lines:#738
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function getTimeToClaim(address _staker) public view returns (uint){
```

Lines:#748
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function claimDivs() public  returns (bool){
```

Lines:#755
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function getStakersList(uint startIndex, uint endIndex) public view returns
(address[] memory stakers, uint[] memory stakingTimestamps,  uint[] memory
lastClaimedTimeStamps, uint[] memory stakedTokens) {
```

Lines:#779
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function transferAnyERC20Tokens(address _tokenAddr, address _to, uint _amount)
public onlyOwner  returns (bool){
```

Lines:#785
@ BSB_StakingStrongStakers;BSB_StakingSoftStakers;BSB_StakingSmartStakers

```
function getClaimableAmount(address account) public view returns (uint){
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **1** high and **4** low severity issues.

Notice:

A lot of loops could affect the functions "End" (high gas fee) and "getTotalRewards (unavailability) as soon as many users will do their stakes.

Customer acknowledged of the possible functions dysfunction (End [high gas fee], getTotalRewards [unavailibility]) because of using cycles over the stakeholders array.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.