QuillAudits

Audit Report
June, 2021

paybswap

Contents

Audit Details and Target

1. Contract

https://drive.google.com/file/d/1whRVtOQW982R_xpEhh6NJSSknssVilX2/view?usp=sharing

2. Audit Target

- To find the security bugs and issues regarding security, potential risks, and critical bugs.
- Check gas optimisation and check gas consumption.
- Check function reusability and code optimisation.
- Test the limit for token transfer and check the accuracy in decimals.
- Check the functions and naming conventions.
- Check the code for proper checks before every function call.
- Event trigger checks for security and logs.
- Checks for constant and function visibility and dependencies.
- Validate the standard functions and checks.
- Check the business logic and correct implementation.
- Automated script testing for multiple use cases including transfers, including values and multi transfer check.
- Automated script testing to check the validations and limit tests before every function call.
- Check the use of data type and storage optimisation.
- Calculation checks for different use cases based on the transaction, calculations and reflected values.

Functions list and audit details

Major Functions

- withdrawBNBer()
- withdrawTokens()
- checkHowManyOwners()
- deleteOperation()

- recieveTokens()
- writeTransaction()
- sendTokens()
- onlySomeOwners()
- transferOwnership()
- transferOwnershipWithHo
- wMany()
- OwnershipTransferred()

Overview of The Contract

Paybswap contract is a token transfer, lock, swap and migration tool.
The token contract is owned by multi owners to verify and validate transactions and approve the swap. Contract functions are owned by owners and used to transfer based on offline calculations, and tokens are input based on the input values.
The token contract is erc20 and bep20 based contract standards for managing user tokens.
Both directories have similar code based on blockchain protocols and standards.
The contract is designed keeping in mind the pay and swap business logic with validating and whitelisting the user transaction details.

Tokenomics

As per the information provided, the tokens generated will be initially transferred to the contract owner and then further division will be done based on the business logic of the application. Tokens cannot be directly purchased from the smart contract, so there will be an additional or third-party platform that will help users to purchase the tokens. Tokens can be held, transferred and delegated freely. Tokens are generated based on the fixed flow in the supply, which can be checked by total supply.

Scope of Audit

The scope of this audit was to analyse Paybswap smart contracts codebase for quality, security, and correctness.

Checked Vulnerabilities

The smart contract is scanned and checked for multiple types of possible bugs and issues. This mainly focuses on issues regarding security, attacks, mathematical errors, logical and business logic issues. Here are some of the commonly known vulnerabilities that are considered:

- TimeStamp dependencies.
- Variable and overflow
- Calculations and checks
- SHA values checks
- Vulnerabilities check for use case
- Standard function checks
- Checks for functions and required checks
- Gas optimisations and utilisation
- Check for token values after transfer
- Proper value updates for decimals
- Array checks
- Safemath checks
- Variable visibility and checks
- Error handling and crash issues
- Code length and function failure check
- Check for negative cases
- D-DOS attacks
- Comments and relevance
- Address hardcoded
- Modifiers check
- Library function use check
- Throw and inline assembly functions
- Locking and unlocking (if any)
- Ownable functions and transfer ownership
- checksArray and integer overflow possibility checks
- Revert or Rollback transactions check

Techniques and Methods

- Manual testing for each and every test cases for all functions.
- Running the functions, getting the outputs and verifying manually for multiple test cases.
- Automated script to check the values and functions based on automated test cases written in JS frameworks.
- Checking standard function and check compatibility with multiple wallets and platforms
- Checks with negative and positive test cases.
- Checks for multiple transactions at the same time and checks d-dos attacks.
- Validating multiple addresses for transactions and validating the results in managed excel.
- Get the details of failed and success cases and compare them with the expected output.
- Verifying gas usage and consumption and comparing with other standard token platforms and optimizing the results.
- Validate the transactions before sending and test the possibilities of attacks.

Structural Analysis
In this step we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.
SmartCheck.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified.

Static Analysis
Static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step a series of automated tools are used to test security of smart contracts.

Gas Consumption
In this step we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

High severity issues

Issues that must be fixed before deployment else they can create major issues.

Medium level severity issues

These issues will not create major issues in working but affect the performance of the smart contract.

Low level severity issues

These issues are more suggestions that should be implemented to refine the code in terms of gas, fees, speed and code accuracy

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	0	0
Closed	2	1	3	2

Issues Found – Code Review / Manual Testing

High severity issues

1. **Contract:** BNB/bnbBridge.sol and ETH/bnbBridge.sol
   Line: 23 - 31
   Issues: Failing for most of the tested tokens, tokens transferred will get locked.
   Reasons: Missing approval function before transfer from another contract. The destination address is a token address, so tokens will remain in the contract address only.
   Suggested Fixes: Either add this description for approve or explain.
   Status: Acknowledged by the developer and discussed. (Closed)

```
23      function sendTokens(uint256 amount) public {
24          require(msg.sender != address(0), "Zero account");
25          require(amount > 0,"Amount of tokens should be more then 0");
26          require(token.balanceOf(msg.sender) >= amount,"Not enough balance");
27
28          transferStatus = token.transferFrom(msg.sender, address(this), amount);
29          if (transferStatus == true) {
30              tokensRecieved[msg.sender] += amount;
31          }
32      }
```

2. **Contract:** BNB/bnbBridge.sol and ETH/bnbBridge.sol
   Line: 34-42
   Issues: Sending different types of tokens can be identified, the basis of amount calculation is not described.
   Reasons: No proper description of the function definition, checks are not proper for value, tokens or value type not defined.
   Status: Acknowledged by the developer and fixed. (Closed)

```
34      function writeTransaction(address user, uint256 amount) public onlyAllOwners {
35          require(user != address(0), "Zero account");
36          require(amount > 0,"Amount of tokens should be more then 0");
37          require(!avoidReentrancy);
38
39          avoidReentrancy = true;
40          tokensRecievedButNotSent[user] += amount;
41          avoidReentrancy = false;
42      }
```

Medium severity issues

3. **Contract:** BNB/bnbBridge.sol and ETH/bnbBridge.sol
   Line: 44-62
   Issues: Smsg.value is not recorded, commissions are not defined or unstructured.
   Reasons: msg.value needed to be recorded as this will help to check the user amount in future. The commission array structure is not defined, which means users are free to add any amount, and this will be transacted to admins in the same series.
   Status: Acknowledged by the developer and fixed as values not needed to be saved. (Closed).

```
44      function recieveTokens(uint256[] memory commissions) public payable {
45          if (tokensRecievedButNotSent[msg.sender] != 0) {
46              require(commissions.length == owners.length, "The number of commissions and owners doe
47              uint256 sum;
48              for(uint i = 0; i < commissions.length; i++) {
49                  sum += commissions[i];
50              }
51              require(msg.value >= sum, "Not enough BNB (The amount of BNB is less than the amount o
52              require(msg.value >= owners.length * 150000 * 10**9, "Not enough BNB (The amount of BN
53
54              for (uint i = 0; i < owners.length; i++) {
55                  address payable owner = payable(owners[i]);
56                  uint256 commission = commissions[i];
57                  owner.transfer(commission);
58              }
59
60              amountToSent = tokensRecievedButNotSent[msg.sender] - tokensSent[msg.sender];
61              token.transfer(msg.sender, amountToSent);
62              tokensSent[msg.sender] += amountToSent;
63          }
64      }
```

Low level severity issues

4. **Contract:** BNB/multiOwnable.sol and ETH/multiOwnable.sol
   Line: 224-230
   Issues: Deleting all ownersIndices will delete all the entries and affect the dependent functions.
   Reasons: ownersIndices is being used by many functions whose value will mismatch at the time of checking. New entries will mismatch the value in ownersIndices.
   Status: Acknowledged by the developer and as it does not need to be managed. (Closed)

```
224         for (uint j = 0; j < owners.length; j++) {
225             delete ownersIndices[owners[j]];
226         }
227         for (uint i = 0; i < newOwners.length; i++) {
228             require(newOwners[i] != address(0), "transferOwnershipWithHowMany
229             require(ownersIndices[newOwners[i]] == 0, "transferOwnershipWithHo
230             ownersIndices[newOwners[i]] = i + 1;
231         }
```

5. **Contract:** BNB/multiOwnable.sol and ETH/multiOwnable.sol
   Line: 191-198
   Issues: Excessive gas usage
   Reasons: Exponential gas usage in this function, as checked in many test cases and scenarios
   Status: Acknowledged by the developer and as it will work for the use case. (Closed)

```
190     */
191     function cancelPending(bytes32 operation) public onlyAnyOwner {
192         uint ownerIndex = ownersIndices[msg.sender] - 1;
193         require((votesMaskByOperation[operation] & (2 ** ownerIndex)) != 0, "cance
194         votesMaskByOperation[operation] &= ~(2 ** ownerIndex);
195         uint operationVotesCount = votesCountByOperation[operation] - 1;
196         votesCountByOperation[operation] = operationVotesCount;
197         emit OperationDownvoted(operation, operationVotesCount, owners.length, msg
198         if (operationVotesCount == 0) {
```

6. **Contract:** BNB/bnbBridge.sol and ETH/bnbBridge.sol
   Line: 66-80
   Issues: Failing for transferred tokens on line 71
   Reasons: On line 71, tokens are not specified or approved, which lead to transaction failure. Tested for multiple tokens but falling. Logic not specified for calculation of this amount on withdrawTokens function.
   Status: Acknowledged and this issue is now marked as closed. (Closed)

```
66      function withdrawTokens(uint256 amount, address reciever) public onlyAllOwners {
67          require(amount > 0,"Amount of tokens should be more then 0");
68          require(reciever != address(0), "Zero account");
69          require(token.balanceOf(address(this)) >= amount,"Not enough balance");
70
71          token.transfer(reciever, amount);
72      }
73
74      function withdrawBNBer(uint256 amount, address payable reciever) public onlyAllOwners {
75          require(amount > 0,"Amount of tokens should be more then 0");
76          require(reciever != address(0), "Zero account");
77          require(address(this).balance >= amount,"Not enough balance");
78
79          reciever.transfer(amount);
80      }
```

Informational

7. The calculation of funds and tokens is not transparent. Token details are not mentioned in the swap contract.

   Status: Acknowledged and this issue is now marked as closed as info is sufficient. (Closed)

8. Missing approve and events loggers.

   Status: Acknowledged and this issue is now marked as closed as not needed. (Closed)

Functional Test Table

Function Names	Technical results	Logical results	Overall
transfer()	Pass	Pass	Pass
transferFrom()	Pass	Pass	Pass
withdrawBNBer()	Pass	Pass	Pass
withdrawTokens()	Pass	Revised	Pass
recieveTokens()	Pass	Revised	Pass
writeTransaction()	Pass	Revised	Pass
sendTokens()	Pass	Revised	Pass
onlySomeOwners()	Pass	Pass	Pass
checkHowManyOwners()	Pass	Revised	Pass
deleteOperation()	Pass	Pass	Pass
transferOwnership()	Pass	Pass	Pass
transferOwnershipWithHowMany()	Pass	Pass	Pass
OwnershipTransferred()	Pass	Pass	Pass
approve()	Pass	Pass	Pass
decreaseAllowance()	Pass	Pass	Pass
increaseAllowance()	Pass	Pass	Pass

Closing Summary

Apart from the issue and suggestion, other contract functions are working fine as most of the functions are standard functions and mostly used independently for multiple contracts. The main business logic of the swap contract is failing in some test cases, which needs to be fixed before the actual implementation.

Code comments are not proper, gas usage of some functions are exponential, which should be minimised as this will be an issue in the case of ethereum.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the Paybswap platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the Paybswap Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

paybswap

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com