QuillAudits

Audit Report
September, 2021

For

paybswap

Contents

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the PaybSwap smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

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

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis
In this step, we have analysed 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.

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 the security of smart contracts.

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

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 the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
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

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	2	0	0
Closed	0	1	18	1

Introduction

During the period of **August 25, 2021 to September 09, 2021** - QuillAudits Team performed a security audit for **PaybSwap** smart contracts.

The code for the audit was taken from the following official repo of PaybSwap:
https://github.com/PaybSwapp/smart-contract/commit/001efa04b9331391dad321ee3ef05b5cca9637ca

Version	Date	Commit hash
1	August	001efa04b9331391dad321ee3ef05b5cca9637
2	September	5cb5db5ade06042b0d01cd0f312c0873d7ab6aee
3	September	d52b16c997f8055ec6f2445dcb3617e1de44f76c
4	September	9599e3f6e74c4005a653ad11d38a8c6a01393ae2

Issues Found

A. Contract – BSCBridgeContract

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

A.1 Floating pragma

```
Line 1:
pragma solidity ^0.6.0;
```

Description
The contract makes use of the floating-point pragma 0.6.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Fixed
The PaybSwap team has fixed the issue in version 2 by locking the pragma version to 0.6.12.

A.2 Missing address verification

```
Line 39:
function changeGovernance(address _governance) public onlyOwner {
        governance = _governance;
        }
```

Description

Certain functions lack a safety check in the address. The address-type argument should include a zero-address test; otherwise, the contract's functionality may become inaccessible, or tokens may be burned in perpetuity.

Remediation

It is recommended to add a require or modifier in order to verify that the input address is different from the zero-address.

Fixed

The PaybSwap team has fixed the issue in version 2 by adding a require in order to verify that the _governance argument is different from the zero-address.

B. Contract – ETHBridgeContract

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

B.1 Floating pragma

```
Line 1:
pragma solidity ^0.6.0;
```

Description
The contract makes use of the floating-point pragma 0.6.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Fixed
The PaybSwap team has fixed the issue in version 2 by locking the pragma version to 0.6.12.

B.2 Missing address verification

```
function changeGovernance(address _governance) public onlyOwner {
        governance = _governance;
        }
```

Description

Certain functions lack a safety check in the address; the address-type argument should include a zero-address test; otherwise, the contract's functionality may become inaccessible, or tokens may be burned in perpetuity.

Remediation

It is recommended to add a require or modifier in order to verify that the input address is different from the zero-address.

Fixed

The PaybSwap team has fixed the issue in version 2 by adding a require to make sure that the _governance argument is different from the zero-address.

C. Contract – Controller

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

C.1 Floating pragma

```
Line 1:
pragma solidity ^0.8.0;
```

Description
The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps to ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Fixed
The PaybSwap team has fixed the issue in version 2 by locking the pragma version to 0.6.12.

C.2 Missing address verification

```
Line 36:
function setRewards(address _rewards) public {
        require(msg.sender == governance, "!governance");
        rewards = _rewards;
        }
```

```
Line 41:
function setStrategist(address _strategist) public {
        require(msg.sender == governance, "!governance");
        strategist = _strategist;
        }
```

```
Line 51:
function setOneSplit(address _onesplit) public {
        require(msg.sender == governance, "!governance");
        onesplit = _onesplit;
        }
```

```
Line 56:
function setGovernance(address _governance) public {
        require(msg.sender == governance, "!governance");
        governance = _governance;
        }
```

Description

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or tokens may be burned in perpetuity.

Remediation

It is recommended to add a require or modifier in order to verify that the input address is different from the zero-address.

Fixed

The PaybSwap team has fixed the issue in version 3 by adding require statements in the previous functions in order to verify that the address argument is different from the zero-address.

C.3 Missing value verification

```
Line 46:
function setSplit(uint256 _split) public {
        require(msg.sender == governance, "!governance");
        split = _split;
        }
```

Description

In the setSplit function, there is a missing verification to the value of the _split argument; this value should be less than the max variable.

Remediation

It is recommended to add a require or modifier in order to verify that the __split argument is lower than max.

Fixed

The PaybSwap team has fixed the issue in version 3 by adding require statements in the previous functions in order to verify that the split variable is less than MAX.

D. Contract – PayBSwapRegistry

High severity issues

No issues were found.

Medium severity issues

D.1 For loop over dynamic array

```
Line 175:
function getVaults() external view returns (address[] memory) {
        address[] memory vaultsArray = new address[](vaults.length());
        for (uint256 i = 0; i < vaults.length(); i++) {
            vaultsArray[i] = vaults.at(i);
            }
        return vaultsArray;
        }
```

```
Line 193:
function getVaultsInfo() external view returns (
            address[] memory vaultsAddresses,
            address[] memory controllerArray,
            address[] memory tokenArray,
            address[] memory strategyArray,
            bool[] memory isWrappedArray,
            bool[] memory isDelegatedArray
        )
        {
        vaultsAddresses = new address[](vaults.length());
        controllerArray = new address[](vaults.length());
        tokenArray = new address[](vaults.length());
        strategyArray = new address[](vaults.length());
        isWrappedArray = new bool[](vaults.length());
        isDelegatedArray = new bool[](vaults.length());
        for (uint256 i = 0; i < vaults.length(); i++) {
            vaultsAddresses[i] = vaults.at(i);
            (address _controller, address _token, address _strategy,
                bool _isWrapped,bool _isDelegated) = getVaultData(vaults.at(i));
            controllerArray[i] = _controller;
            tokenArray[i] = _token;
            strategyArray[i] = _strategy;
            isWrappedArray[i] = _isWrapped;
            isDelegatedArray[i] = _isDelegated;
        }
        }
```

Description
When smart contracts are deployed, or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation
Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Risk acknowledged
The PaybSwap team has acknowledged the risk.

Low level severity issues

D.2 Floating pragma

```
Line 3:
pragma solidity ^0.8.0;
```

Description
The contract makes use of the floating-point pragma 0.8.0. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps to ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Fixed
The PaybSwap team has fixed the issue in version 2 by locking the pragma version to 0.6.12.

D.3 pendingGovernance not set back to address (0)

```
Line 248:
function acceptGovernance() external onlyPendingGovernance {
        governance = msg.sender;
        }
```

Description
In the acceptGovernance function, after the msg.sender becomes the owner, the pendingGovernance is still set to the old value, but it should be set back to the address(0).

Remediation
The pendingGovernance should be set back to the address (0).

Fixed
The PaybSwap team has fixed in version 3 the issue by setting the address of the pendingGovernance back to 0

Informational Issues

D.4 Comments Left In The Code

```
Line 143:
  require(controllerVault == vault,"Controller vault address does not match");
      // Might happen on Proxy Vaults

      // Check if strategy has the same token as vault
      //       if (isWrapped) {
      //           address underlying = IVault(vault).underlying();
      //           require(underlying == token, "WrappedVault token address does not match");
      //           // Might happen?
      //       } else if (!isDelegated) {
      //           address strategyToken = IStrategy(strategy).want();
      //           require(token == strategyToken, "Strategy token address does not match");
      //           // Might happen?
      //       }

      return (controller, token, strategy, isWrapped, isDelegated);
```

Description
There is still a commented unused code in the smart contract.

Remediation
The pendingGovernance should be set back to the address (0).

Fixed
The PaybSwap team has fixed the issue in version 2 by uncommenting the code.

E. Contract – PaybNftMarketplace

High severity issues

No issues were found.

Medium severity issues

E.1 For loop over dynamic array

```
function getAllTokensObjs() public view returns (Token[] memory) {
        Token[] memory _tokensObj = new Token[](tokensCount);
        uint256 _id = 1;
        while (_owners[_id] != address(0)) {
            Token memory _token = Token({
                id: _id,
                price: prices[_id],
                token: tokenAddress[_id],
                owner: _owners[_id],
                uri: _tokenURIs[_id],
                status: isForSale(_id) });
            _tokensObj[_id - 1] = _token;
            _id++;
        }
        return _tokensObj;
    }
```

Description
When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial-of-Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation
Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Risk acknowledged
The PaybSwap team has acknowledged the risk.

E.2 The use of transfer instead of safeTransfer

```
Line 42:
function withdraw() external onlyOwner {
        owner.transfer(getBalance());
        }
```

Description
The ERC20 standard token implementation functions also return the transaction status as a Boolean. It's good practice to check for the return status of the function call to ensure that the transaction was successful. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails. However, it is mostly missed by developers when they carry out checks; in effect, the transaction would always succeed, even if the token transfer didn't.

Remediation
Implement SafeERC20 in order to use the safeTransfer function instead of transfer.

Fixed
The PaybSwap team has fixed the issue in version 2 by using safeTransfer instead of transfer.

Low level severity issues

E.3 Floating pragma

```
Line 3:
pragma solidity >=0.8.4 <=0.8.6;
```

Description

The contract makes use of the floating-point pragma 0.8.4< X <0.8.6. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps to ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation

Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Fixed

The PaybSwap team has fixed the issue in version 2 by locking the pragma version to 0.8.4.

E.4 Missing address verification

```
Line 46:
function withdraw(address _user, uint256 _amount) external onlyOwner {
        uint256 _balance = getBalance();
        require(balance >= _amount,
            "Amount is greater than the balance of contract"
        );
        payable(_user).transfer(_amount);
 }
```

```
Line 56:
function withdraw(address _tokenErc20, address _user) external onlyOwner {
        uint256 _totalBalance = IERC20(_tokenErc20).balanceOf(address(this));
        require(_totalBalance > 0, "Total balance is null");
        IERC20(_tokenErc20).transfer(_user, _totalBalance);
        }
```

```
function sendToAdmin(address _token, address _admin, uint256 _amount)
public payable {
        require(
            _token.isContract() || _token == address(0),
            "Token isn't a contract"
        );
        if (_token == address(0)) {
            payable(_admin).transfer(msg.value);
        } else {
            require(
                IERC20(_token).balanceOf(msg.sender) >= _amount,
                "Insufficient funds"
            );
            IERC20(_token).transferFrom(msg.sender, _admin, _amount);
        }
    }
```

Description

Certain functions lack a safety check in the address; the address-type argument should include a zero-address test. Otherwise, the contract's functionality may become inaccessible, or tokens may be burned in perpetuity.

Remediation

It is recommended to add a require or modifier in order to verify that the input address is different from the zero-address in withdraw function and if it's really the admin's address in the sendToAdmin function.

Fixed

The PaybSwap team has fixed the issue in version 2 by adding require statements in the previous functions in order to verify that the address argument is different from the zero-address.

E.5 Missing value verification

```
Line 231:
    function setNewRoyalties(uint256 _tokenId, uint256 _newRoyalties) public
      tokenNotFound(_tokenId)
    {
    require(msg.sender == ownerOf(_tokenId),"Sender isn`t the owner of token");
      require(_newRoyalties > royalties[_tokenId],
        "New royalties cannot be less than the old one");
      royalties[_tokenId] = _newRoyalties;
      emit RoyaltiesChanged(_tokenId, _newRoyalties, msg.sender);
    }
```

Description

In the setNewRoyalties function there is a missing verification in the value of the _newRoyalties argument; this value should be less than 100.

Remediation

The Team PaybSwap should add a require or modifier in order to verify that the _newRoyalties argument is less than 100.

Fixed

The team PaybSwap has fixed this issue in version 3 by adding a modifier to verify that _newRoyalities is less than 100.

F. Contract – StakeRewardPoolPAYB

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

F.1 Floating pragma

```
Line 1:
pragma solidity ^0.6.0;
```

Description
The contract makes use of the floating-point pragma 0.7.6. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps to ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Fixed
The PaybSwap team has fixed the issue in version 2 by locking the pragma version to 0.6.12.

F.2 Comparing the curent time with an already passed time

```
function stake(uint256 amount) public override updateReward(msg.sender) {
        require(amount > 0, "Cannot stake 0");
        require(block.timestamp > 1600790400, "Cannot stake yet");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }
```

Description

In the stake function there is a require that compares implies that the current timestamp is bigger than an already passed date that is 22 Sep 2020. That means that the require is irrelevant, and it will pass every time.

Remediation

The timestamp should be set to a newer date.

Fixed

The PaybSwap team has fixed the issue in version 2 by changing the timestamp to 01.10.2021.

F.3 Comparing the curent time with an already passed time

```
function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }
```

```
function stake(uint256 amount) public override updateReward(msg.sender) {
        require(amount > 0, "Cannot stake 0");
        require(block.timestamp > 1600790400, "Cannot stake yet");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }
```

Description

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners.Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right; all what is guaranteed is that it is higher than the timestamp of the previous block.

Remediation

You can use an Oracle to get the exact time or verify if a delay of 900 seconds won't destroy the logic of the staking contract.

Fixed

The PaybSwap team has accepted the risk knowing that the 900 seconds delay won't impact the contract's logic.

G. Contract – PaybSwapVault

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

G.1 Floating pragma

```
Line 1:
pragma solidity ^0.6.0;
```

Description
The contract makes use of the floating-point pragma 0.7.6. Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps to ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Fixed
The PaybSwap team has fixed the issue in version 2 by locking the pragma version to 0.6.12.

G.2 Missing address verification

```
function setGovernance(address _governance) public {
        require(msg.sender == governance, "!governance");
        governance = _governance;
    }
```

```
function setController(address _controller) public {
        require(msg.sender == governance, "!governance");
        controller = _controller;
    }
```

Description

Certain functions lack a safety check in the address; the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible, or tokens may be burned in perpetuity.

Remediation

It is recommended to add a require or modifier in order to verify that the input address is different from the zero-address.

Fixed

The PaybSwap team has fixed the issue in version 2 by adding require statements in the previous functions in order to verify that the address argument is different from the zero-address.

G.3 Missing value verification

```
Line 37:
function setMin(uint256 _min) external {
        require(msg.sender == governance, "!governance");
        min = _min;
        }
```

Description
In the setMin function the _min argument should be lower than the max variable.

Remediation
It is recommended to add a require to check if the _min argument is lower than the max variable.

Fixed
The paybSwap team fixed this issue in version 2 using a modifier to verify that _min variable is less than MAX.

Automated Tests

Solidity Static Analysis

Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in PaybNftMarketplace.buy(uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 115:4:

Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in PaybNftMarketplace.sendTo(uint256,address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 262:4:

Check-effects-interaction:
Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 122:4:

Inline assembly:
The Contract uses inline assembly, this is only advised in rare cases.
Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 263:20:

Inline assembly:
The Contract uses inline assembly, this is only advised in rare cases.
Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 32:8:

Inline assembly:
The Contract uses inline assembly, this is only advised in rare cases.
Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 201:16:

Low level calls:
Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 57:27:

Low level calls:
Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 131:50:

Low level calls:
Use of "delegatecall": should be avoided whenever possible.
External code, that is called can change the state of the calling contract and send ether from the caller's balance.
If this is wanted behaviour, use the Solidity library feature if possible.
more
Pos: 185:50:

This on local calls:

Use of "this" for local functions: Never use "this" to call functions in the same contract, it only consumes more gas than normal local calls.
more
Pos: 168:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.
Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 235:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.
Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 405:8:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.
Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 443:8:

Mythx

Issues

These were the issues detected in the scan. You can use the toggles to filter by severity and display or hide ignored issues.

Severity: Low (3) | Medium (0) | High (0)

Ignored Issues: Hidden (0)

ID	Severity	Name	File	Location
SWC-103	Low	A floating pragma is set.	SafeMath.sol	L: 1 C: 0
SWC-108	Low	State variable visibility is not set.	SafeMath.sol	L: 10 C: 31
SWC-108	Low	State variable visibility is not set.	SafeMath.sol	L: 10 C: 73

Issues

These were the issues detected in the scan. You can use the toggles to filter by severity and display or hide ignored issues.

Severity: Low (6) | Medium (0) | High (0)

Ignored Issues: Hidden (0)

ID	Severity	Name	File	Location
SWC-103	Low	A floating pragma is set.	IERC20.sol	L: 2 C: 0
SWC-108	Low	State variable visibility is not set.	IERC20.sol	L: 24 C: 0
SWC-108	Low	State variable visibility is not set.	IERC20.sol	L: 26 C: 5
SWC-108	Low	State variable visibility is not set.	IERC20.sol	L: 26 C: 46
SWC-108	Low	State variable visibility is not set.	IERC20.sol	L: 29 C: 2
SWC-108	Low	State variable visibility is not set.	IERC20.sol	L: 29 C: 34

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.

Many issues were discovered during the initial audit; the PaybSwap team fixed all the issues discovered in our Audit.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **PaybSwap Contract**. 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.

Audit Report September, 2021

For

paybswap

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com