QuillAudits

Audit Report
Setptember, 2021

Contents

Scope of Audit

The scope of this audit was to analyze and document the PapayaSwap 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 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.

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.

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.

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

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

High severity issues

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 to be fixed before moving to a live environment.

Medium level severity issues

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 level severity issues

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 severity four issues which 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	0	0	0
Closed	0	0	0	0

Introduction

During the period of **September 6, 2021 to September 7, 2021** - QuillAudits Team performed a security audit for the PapayaSwap smart contract.

The code for the audit was taken from following the official link: https://bscscan.com/address/0xc6604d3e9dc6494098dab71e56a4518344d9825a#code

Issues Found – Code Review / Manual Testing

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low level severity issues

No issues were found.

Informational

No issues were found.

Functional test

Function Names	Testing results
constructor	PASS
getOwner	PASS
decimals	PASS
symbol	PASS
name	PASS
totalSupply	PASS
balanceOf	PASS
Transfer	PASS
allowance	PASS
approve	PASS
transferFrom	PASS
increaseAllowance	PASS
decreaseAllowance	PASS
mint	PASS

Automated Testing

Slither

```
INFO:Detectors:
BEP20Token.allowance(address,address).owner (contracts/PapayaSwap.sol#423) shadows:
        - Ownable.owner() (contracts/PapayaSwap.sol#301-303) (function)
BEP20Token._approve(address,address,uint256).owner (contracts/PapayaSwap.sol#578) shadows:
        - Ownable.owner() (contracts/PapayaSwap.sol#301-303) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
BEP20Token._burn(address,uint256) (contracts/PapayaSwap.sol#557-563) is never used and should be removed
BEP20Token._burnFrom(address,uint256) (contracts/PapayaSwap.sol#592-595) is never used and should be removed
Context._msgData() (contracts/PapayaSwap.sol#117-120) is never used and should be removed
SafeMath.div(uint256,uint256) (contracts/PapayaSwap.sol#216-218) is never used and should be removed
SafeMath.div(uint256,uint256,string) (contracts/PapayaSwap.sol#231-238) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/PapayaSwap.sol#251-253) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/PapayaSwap.sol#266-269) is never used and should be removed
SafeMath.mul(uint256,uint256) (contracts/PapayaSwap.sol#191-203) is never used and should be removed
SafeMath.sub(uint256,uint256) (contracts/PapayaSwap.sol#162-164) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Redundant expression "this (contracts/PapayaSwap.sol#118)" inContext (contracts/PapayaSwap.sol#108-121)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
BEP20Token.constructor() (contracts/PapayaSwap.sol#355-363) uses literals with too many digits:
        - _totalSupply = 20000000000 * 10 ** 18 (contracts/PapayaSwap.sol#359)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (contracts/PapayaSwap.sol#320-323)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (contracts/PapayaSwap.sol#329-331)
increaseAllowance(address,uint256) should be declared external:
        - BEP20Token.increaseAllowance(address,uint256) (contracts/PapayaSwap.sol#469-472)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20Token.decreaseAllowance(address,uint256) (contracts/PapayaSwap.sol#488-491)
mint(uint256) should be declared external:
        - BEP20Token.mint(uint256) (contracts/PapayaSwap.sol#501-504)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (5 contracts with 75 detectors), 18 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

SOLIDITY STATIC ANALYSIS

Gas costs:

Gas requirement of function BEP20Token.transferOwnership is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 329:2:

Gas costs:

Gas requirement of function BEP20Token.symbol is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 382:2:

Gas costs:

Gas requirement of function BEP20Token.name is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 389:2:

Gas costs:

Gas requirement of function BEP20Token.transfer is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 415:2:

Gas costs:

Gas requirement of function BEP20Token.approve is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 434:2:

Gas costs:

Gas requirement of function BEP20Token.transferFrom is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 451:2:

Gas costs:

Gas requirement of function BEP20Token.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 469:2:

Gas costs:

Gas requirement of function BEP20Token.decreaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 488:2:

Gas costs:

Gas requirement of function BEP20Token.mint is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 501:2:

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 16:2:

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 375:2:

Miscellaneous

Constant/View/Pure functions:

SafeMath.sub(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 162:2:

Constant/View/Pure functions:

SafeMath.div(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 216:2:

Constant/View/Pure functions:

SafeMath.mod(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 251:2:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 539:12:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 541:36:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 542:14:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 542:35:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 542:48:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 542:35:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 542:48:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 543:30:

Similar variable names:

BEP20Token._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 543:39:

Similar variable names:

BEP20Token._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 558:12:

Similar variable names:

BEP20Token._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 560:14:

Similar variable names:

BEP20Token._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 560:35:

Similar variable names:

BEP20Token._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 560:48:

Similar variable names:

BEP20Token._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 561:36:

Similar variable names:

BEP20Token._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 562:39:

Similar variable names:

BEP20Token._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 593:10:

Similar variable names:

BEP20Token._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 593:19:

Similar variable names:

BEP20Token._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 594:13:

Similar variable names:

BEP20Token._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 594:48:

Similar variable names:

BEP20Token._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 594:75:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 148:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 176:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 200:4:

SOLIDITY STATIC ANALYSIS

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 521:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 522:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 539:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 558:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 579:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 580:4:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 200:12:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 234:16:

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.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the PapayaSwap 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 PapayaSwap Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Closing Summary

Overall, smart contracts are very well written. No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com