QuillAudits

Audit Report
March, 2021

PROXY

Contents

Scope of Audit

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

Check Vulnerabilities

- 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 contracts 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

	High	Medium	Low	Informational
Open	0	0	0	0
Closed	3	7	5	0

Introduction

DuDuring the period of **March 29th, 2021 to April 4th, 2021** - Quillhash Team performed a security audit for **BTC Proxy** smart contracts.
The code for the audit was taken from following the official GitHub link:

- **BTCpx-ERC20**
  https://github.com/Proxy-Protocol/BTCpx-ERC20/blob/main/contracts/BTCpx.sol

  Commit Hash - 5ca45abbe3d0eb2ee6abf92726b74f44c3a75e58

- **Relay**
  https://github.com/Proxy-Protocol/Matic-Layer2/blob/master/Relay.sol

  Commit Hash - 63fbeeaabf79a336263072a8d4a758a8b02ac7e0

Issues Found – Code Review / Manual Testing

A. Contract Name - BTCpx

Medium severity issues

1. **No Events emitted after imperative State Variable modification**

Status: CLOSED

Description:
Functions that update an imperative arithmetic state variable contract should emit an event after the updation of that variable.
In the BTCpx contract, the following functions modify some crucial arithmetic parameters like **burnFee, mintFee5:**

- **setMintFee()**
- **setBurnFee()**

Since there is no event emitted on updating these variables, it might be difficult to track it off-chain.

Recommendation:
An event should be fired after changing crucial arithmetic state variables.

2. **Zero Address Validation check should be implemented**
   Line no. 63
   Status: CLOSED

   Description:
   In order to eliminate any unwanted scenario where a Zero Address is passed as a function argument, a **require statement** must be implemented.

   The function **setDaoUser at Line 63** should implement a Zero Address Validation check.

   Recommendation:
   The **_addr** argument should be checked with a **require** statement.
   require(_addr != address(0),"Invalid address passed");

Low level severity issues

3. **External visibility should be preferred**
   Status: CLOSED

   Description:
   Those functions that are never called throughout the contract should be marked as **external** visibility instead of **public** visibility.
   This will effectively result in Gas Optimization as well.
   Therefore, the following function must be marked as **external** within the contract:

   - **setData**
   - **setDAOUser**
   - **setDAOUserAccountId**
   - **setMintFee**
   - **setBurnFee**
   - **burn(address,uint256)**
   - **burn(bytes,uint256)**
   - **mint(uint256)**
   - **mint(address,uint256)**
   - **getDAOUserAccountId**
   - **getMintStatus**

Recommendation:
The above-mentioned functions should be assigned external visibility.

4. Function with similar names should be avoided
Line no - 107, 116, 148, 162

Status: CLOSED

Description:
The BTCpx contract includes a few functions with exactly similar names. Since every function has different behaviour, it is considered a better practice to avoid similar names for 2 different functions.

Mentioned below are the functions with similar names but different behavior and arguments:

- **burn() -> Line 107 and 116**
- **mint() -> 148 and 162**

Recommendation:
The above-mentioned functions should be assigned external visibility.

B. Contract Name - Relay.sol

High severity issues

New Issues Found after Final Audit

1. ProcessTx will work with unconfirmed transactions

Status: CLOSED

Description:
L461-L466 - If the number of confirmations are not enough, we are still adding to the pendingMints mapping and setting txsInformation[uuid].status to false. But we are not checking this status key while processTx is called. While means processing will take irrespective of whether the minimum number of confirmation criteria is met or not.

Recommendation:
Add validation to check if the status is true

2. verifyTx function prone to repeated minting attack

Status: CLOSED

Description:
Function verifyTx takes the parameters in input and adds an entry in the pending mints after validating by calling parseRelayData internally. This function is missing a mapping of BTC transaction Id => boolean which will mark a transaction id as already verified and minted. This will prevent minting for the same transaction id repeatedly.

Recommendation:
Add mapping of :-
mapping(bytes => bool) btcTxProcessed in the Relay Contract
and btcTxProcessed[txid] = true; at the end of parseRelayData by taking in txid as input.

And finally add require(btcTxProcessed[txid]==false, “Transaction already minted”)

3. verifyTx function prone to minting for zero confirmation transactions

Status: CLOSED

Description:
Function verifyTx is prone to adding values to pendingMints mapping for the transactions with zero confirmations. To prevent this function parameter requiredconfirmations should be checked for minimum 6 as per the bitcoin best practices for verifying a transaction.

Recommendation:
Add require statement -
require(requiredconfirmations >= 6 ,”Confirmations too low”); in the verifyTx function

Medium severity issues

4. Loops in the Contract are extremely costly

Status: CLOSED

Description:
Most of the for loops in the Relay contract include state variables like .length of a non-memory array, in the condition of the for loops.
As a result, these state variables consume a lot more extra gas for every iteration of the for loop.
The following functions include such loops at the mentioned lines:

- **submitBlockHeaderBatch** at #Line243
- **_reorgChain** at #Line 283
- **parseRelayData** at #Line457
- **find** at #Line199
- **getPendingMints** at #Line505, 512
- **getPendingTransactions** at #Line530, 537
- **getCompletedTransactions** at #Line548

Recommendation:
Its quite effective to use a local variable instead of a state variable like .length in a loop.
For instance,

```
uint256 local_variable =outs.length
for(uint256 i = 0; i < local_variable; i++) {
        if(compareStringsbyBytes(string(outs[i].script),
string(btcAddress))) {
            amount = amount.add(outs[i].value);
            break;
        }
    }
```

5. verifyTx(L-390) and find (L-454) missing zero address check for ethereum address

Status: CLOSED

Description:
ethAddress parameter is being taken as input but is not checked for zero address

Recommendation:
Add require statement -
require(ethAddress != address(0), “Invalid ethereum address”); in the verifyTx function

6. **processTx will send txProcessed event unchecked**

Status: CLOSED

Description:
processTx will send txProcessed event and set txsInformation[id].transferred to true irrespective if it exists or not.

Recommendation:
Add require statement -
require(txsInformation[id].txhex.length > 0 && txsInformation[id].transferred == false , “Transaction not verified yet or already processed”)

7. **removeByIndex and find may result in removing different pendingMints than target pendingMints**

Status: CLOSED

Description:
Find does return an index of the pending mint by searching by value for an ethereum address, but by the time it would try to delete by calling removeByIndex, the index may have already shifted due to any previous removeByIndex call that may have been executed in this meantime.

Recommendation:
A function like findAndRemoveByIndex that will find the index and pass on to the logic to delete the value of penfingMints at that index would work fine since both these things would be happening in the same pass.

8. **removeByIndex and find may result in removing different pendingMints than target pendingMints**

Status: CLOSED

Description:

a) height - 1 + requiredconfirmations <= _currHeight - Line - 405-407
b) height - 1 + requiredconfirmations > _currHeight - Line - 402
In the above code, the current block height check is invalid unless you want one block as a buffer.

Recommendation:

height + requiredconfirmations <= _currHeight - Line - 405-407
And
height + requiredconfirmations > _currHeight - Line - 402

Low level severity issues

New Issues Found after Final Audit

9. parseRelayData gas cost saving

Status: CLOSED

Description:

L443 - We are storing the relaydata in a mapping. If our aim is to prevent adding the transaction twice and to prevent it from getting processed twice in the
parseRelayData function, we may achieve this by storing the txid since a transaction id can be processed once for a single mint process. By this, we can save gas since relaydata will be huge data

10. Comparison to boolean Constant

Line no: 46

Status: CLOSED

Description:

Boolean constants can directly be used in conditional statements or require statements.
Therefore, it's not considered a better practice to explicitly use **TRUE or FALSE** for comparisons.

11. _currHeight and _currBlock may not be needed

Status: CLOSED

Description:
Mostly _currHeight and _currBlock will be the same as _bestBlock and _bestHeight.

Recommendation:
Removal of this will result in gas cost savings.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. During the process of audit, several issues of high, medium as well as low severity were found which might affect the intended behaviour of the contracts.

However, all the issues have now been fixed and checked.

Disclaimer

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

PROXY

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com