BTC Proxy Smart Contract Initial Audit Report

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.

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 analyzed, 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 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 to 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 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 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 four severity 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	2	9	5	3
CLOSED	0	0	0	0

Introduction

During 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

- **PRXY**

  https://github.com/Proxy-Protocol/PRXY/tree/main/contracts

  Commit Hash - e535d1a2a78b6401ea0538a0f2e38d9c8cc9e281

- **Relay**

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

  Commit Hash - 63fbeeaabf79a336263072a8d4a758a8b02ac7e0

A. Contract Name - BTCpx

Issues Found – Code Review / Manual Testing

High Severity Issues
Not Found

Medium Severity Issues

A.1 No Events emitted after imperative State Variable modification
Line no:
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.

A.2 Zero Address Validation check should be implemented
Line no: 63
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 Severity Issues

A.3 External visibility should be preferred

Line no:

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***
- ***getEthAddress***

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

A.4 Function with similar names should be avoided

Line no - 107, 116, 148, 162

Description:
The BTCpx contract includes a few functions with exactly similar names. Since every function has different behavior, it is considered a better practise 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***

Recommended:
It is recommended to avoid using a similar name for different functions.

B. Contract Name - ProxyCoin

Medium Severity Issues

B.1 No Events emitted after imperative State Variable modification

Line no:

Description:

Functions that update an imperative arithmetic state variable contract should emit an event after the updation of that variable.

In the ProxyCoin contract, the following functions modify some crucial arithmetic parameters like ***mintCycleCap, mintDuration:***

- ***changeMinCycleCap()***
- ***changeMintDuration()***

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.

Low Severity Issues

B.2 External visibility should be preferred

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:

- ***startMinting()***
- ***mint()***
- ***changeMintingCycleCap()***
- ***changeMintDuration()***

Recommendation:

The above-mentioned functions should be assigned external visibility.

B.3 Internal Visibility should be assigned

Line no *- 47,51*

Description:

The ProxyCoin contract includes some functions that are supposed to be used effectively within the contract itself in order to perform some logical operations.

Therefore, such functions must be marked as **internal** within the contract:

- ***getMintingStatus()***
- ***mintingFinished()***

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

Informational

1. Coding Style Issues

Description:
Coding style issues influence code readability and in some cases may lead to bugs in future. Smart Contracts have naming convention, indentation and code layout issues. It's recommended to use the Solidity Style Guide to fix all the issues. Consider following the Solidity guidelines on formatting the code and commenting for all the files. It can improve the overall code quality and readability.

2. Order of layout

Description:
The order of functions as well as the rest of the code layout does not follow solidity style guide.

Layout contract elements in the following order:

a. Pragma statements

b. Import statements

c. Interfaces

d. Libraries

e. Contracts

Inside each contract, library or interface, use the following order:

a. Type declarations

b. State variables

c. Events

d. Functions

Please read following documentation links to understand the correct order: -
https://solidity.readthedocs.io/en/v0.7.6/style-guide.html#order-of-layout -
https://docs.soliditylang.org/en/v0.7.6/style-guide.html#order-of-functions

C. Contract Name - Relay.sol

High Severity Issues

C.1 verifyTx function prone to repeated minting attack - Solved

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”)

C.2 verifyTx function prone to minting for zero confirmation transactions - Solved

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

C.3 blockHeight issue when chain reorgs

Description :

L314-L315 -In this function when the reorg happens, you are updating the _bestBlock and _bestHeight while on verifyTx you are reading from the

_currentHeight. So _currentHeight can be a stale value in case of reorg since when reorg happens you are only updating _bestBlock and _bestHeight.

Recommendation:
If you want to keep 2 variables _currentHeight and _bestHeight , then _currentHeight should be updated in the _chainreog function too

C.4 ProcessTx will work with unconfirmed transactions

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 a validation to check if the status is true

Medium Severity Issues

C.5 Loops in the Contract are extremely costly

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;
```

```
    }
  }
```

C.6 verifyTx(L-390) and find (L-454) missing zero address check for ethereum address - Solved

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

C.7 processTx will send txProcessed event unchecked - Solved

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”)

C.8 removeByIndex and find may result in removing different pendingMints than target pendingMints - Solved

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 meanwhile.

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.

C.9 _currHeight and _currBlock may not be needed - Open

Description

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

Recommendation

Removal of this will result in gas cost savings.

C.10 verifytx block height check invalid - Solved

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 Severity Issues

C. 11 Comparison to boolean Constant

Line no: 46

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.

Recommendation:

The equality to boolean constants must be eradicated from the above-mentioned line.

C.12 parseRelayData gas cost saving

Description

L443 - We are storing the relaydata in a mapping. If our aim is to prevent adding the transaction twice and to prevent it 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

Informational

C.10 Compilation Issues

a) Relay.sol:15:1: Warning: Contract code size exceeds 24576 bytes (a limit introduced in Spurious Dragon). This contract may not be deployable on mainnet. Consider enabling the optimizer (with a low "runs" value!), turning off

revert strings, or using libraries. contract Relay is IRelay, Parser, Ownable { ^ (Relevant source part starts here and spans across multiple lines). - Remix warning compilation.

b) Invalid token return hex - in BytesLib.sol and CryptoUtils.sol

Closing Summary

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

However, 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.

Therefore, it is recommended to go through the above-mentioned issues and fix them to avoid any unexpected scenario during the execution of the contract.

Disclaimer