QuillAudits

Audit Report
August, 2021

PROXY

Contents

Scope of Audit

The scope of this audit was to analyze and document the BTC Proxy Token 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	2	1	1	1
Closed	2	0	2	4

Introduction

During the period of **July 02, 2021 to July 04, 2021** - QuillAudits Team performed a security audit for BTC Proxy smart contracts.

The code for the audit was taken from following the official link:
https://github.com/Proxy-Protocol/BTCpx-ERC20/commits/main/contracts/BTCpx.sol
Branch: master

Note	Date	Commit hash
Version 1	July	c45ec1d4c9cb8baa7e4bbde26e20eb73d616aca8
Version 2	July	e05b3902d56aa9c3c6cf78548d1cbe6b269892e8
Version 3	July	8861f1270d7ead0c6ab1981c4a667566ef5de6d4

Issues Found – Code Review / Manual Testing

High severity issues

1. Bypass the mintFee and burnFee

   Description

   According to the setData, burnByAddress, getWithdrawalBtcAmount and getMintBtcAmount, when these functions are called, the users have to pay mintFee and burnFee which are set to 0.1% and 0.3% respectively.

   The vulnerability happens when getUserBurnFee and getUserMintFee are calculated.

   L211:
   value.sub((value.mul(getUserBurnFee(who))).div(percentageDivider));

   L219:
   value.sub((value.mul(getUserMintFee(who))).div(percentageDivider)).sub(txFee);

   In this case, the percentageDivider = 10000. Because Solidity integer division truncates, thus, if the value*getUserBurnFee(who) or value*getUserMintFee(who) is smaller than 10000, the value*getUserBurnFee(who) = Zero and value*getUserMintFee(who) = Zero. Whis also means that the receivers would not be charged for any mintFee or burnFee.

   In conclusion, if the value*getUserBurnFee(who) or value*getUserMintFee(who) is smaller than 10000, the actual minFee and burnFee that the users have to pay is Zero.

   Remediation
   Consider implementing if statements where appropriate to validate all user-controlled input, including constructor, to avoid the potential for erroneous values to result in unexpected behaviors or wasted gas.

   Status: Fixed
   This issue was found fixed in version 3.

2. burnFee is not applied when burnByAddress() is called

Line	Code

131	function burnByAddress(bytes memory _btcAddr, uint256 _amount)
	public virtual {
	_burn(_msgSender(), _amount);
	uint256 _btcAmount = getWithdrawalBtcAmount(_msgSender(),
	_amount);
	emit Withdrawn(_btcAddr, _btcAmount);
	}

Description

As mentioned in the BTC Proxy document, users have to pay some fees when burning. But in this case, when users burn their own _amount using the function burnByAddress, they do not have to pay any burning fees.

This occurs because the user's controllable _amount is used directly for the _burn() function rather than the _btcAmount, whereas the _btcAmount is calculated by the getWithdrawalBtcAmount() function after the burning fees is applied.

As a result, users are not charged for burning fees, and the _btcAmount is useless in this case.

Remediation

The _btcAmount should be used for the _burn() function instead of the _amount

Status: Acknowledged by the Auditee

The Auditee has confirmed that this is not an issue as burn fee is applied when sending btc to users wallet. Thus, it's applied to the amount which is burned and we are burning full btcpx.

3. setDAOUserFees() is not applied when users are called via setData()

Line	Code

131	function mint(address _addr, uint256 _amount, uint256 _txFee, uint256 _uuid) internal virtual {
	_amount = getMintBtcAmount(_msgSender(), _amount, _txFee);
	_mint(_addr, _amount);
	mintStatus[_uuid] = true;
	emit Mint(_addr, _amount, _uuid);
	}

Description
A user can be minted via the setData, which is only callable by Predicate. But in this case, the getMintBtcAmount() function parses the _msgSender() to calculate the mintFee instead of the address in _relayData.

As a result, if a user has already been set for a mintFee externally via setDAOUserFees() function, the user will not be charged for that mintFee; instead, the default value of mintFee will be charged because the getMintBtcAmount() function will parse the _msgSender() which is the Predicate not the user itself.

Remediation
We recommend replacing the variable _msgSender() by _addr at line 133 in the getMintBtcAmount() function.

Status: Fixed
This issue was found fixed in version 2.

4. Bypass the burnFee by calling burn().

Description
The ERC20BurnableUpgradeable.sol contract is being utilized, whereby the burn() function is callable by any users without being charged by the burning fees.

Remediation
Consider utilizing the ERC20Upgradeable.sol contract instead of ERC20BurnableUpgradeable.

Status: Acknowledged by the Auditee
The Auditee has clarified that this is not an issue as the user is burning their btcpx directly to bring scarcity. If any user is calling this function, then they will burn their tokens but won't receive the BTC corresponding to it.

Medium severity issues

5. Centralization Risks

Description
The role owner has the authority to
update critical settings (setDAOUserFees, setDAOUserAccountId, setMintFee, setBurnFee)

Remediation
We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

- with reasonable latency for community awareness on privileged operations;
- Multisig with community-voted 3rd-party independent co-signers;
- DAO or Governance module increasing transparency and community Involvement;

Status: Acknowledged by the Auditee

Low level severity issues

6. Missing Range Check for Input Variables

Description
The role can set the following state variables arbitrary large or small causing potential risks in fees and anti whale :
- _burnFee
- _mintFee

Remediation
We recommend setting ranges and check the following input variables where it's set by any functions:
- burnFee
- mintFee

Status: Fixed
This issue was found fixed in version 2.

7. Missing zero address validation

Line	Code

42	function initialize (address _predicate) public initializer {
	predicate = _predicate;
	__Ownable_init();
	__ERC20_init("Bitcoin Proxy", "BTCpx");
	__ERC20Burnable_init();
	}

49	function setPredicate(address _predicate) public onlyOwner {
	emit PredicateChanged(predicate, _predicate);
	predicate = _predicate;
	}

Description
We've detected missing zero address validation for _predicate address in the aforementioned functions above.

Remediation
Consider implementing require statements where appropriate to validate all user-controlled input, including constructor, to avoid the potential for erroneous values to result in unexpected behaviors or wasted gas.

Status: Closed
This issue was found fixed in version 2.

8. State Variable Default Visibility

Line	Code

95	function setDAOUserAccountId(address _addr, string memory _accountId) external onlyOwner {
	daoUsers[_addr].accountId = _accountId;
	}

Description
Users can have the same _accountId when it's set by the setDAOUserAccountId() function.

Remediation
Please clarify the purpose of the accountId for the users in this contract. We've found that, for all users, the _accountId is not unique. According to the function of the accountId, it should be unique to identify the accounts in a system.

Status: Acknowledged by the Auditee
The Auditee has confirmed that users can have the same _accountId Account id has been used to map the user address corresponding to sub account.

Informational

9. Different pragma directives are used

Pragma version ^0.8.0
Pragma version ^0.8.4

Description
We found that different Solidity versions are used.

Remediation
Use one Solidity version.

Status: Acknowledged by the Auditee
The contract will be deployed with pragma version 0.8.4.

10. State Variable Default Visibility

Line	Code
23	uint256 public constant percentageDivider = 10000;

Description

Constants should be named with all capital letters with underscores separating words. Examples: MAX_BLOCKS, TOKEN_NAME, TOKEN_TICKER, CONTRACT_VERSION

Remediation
Follow the Solidity naming convention.

Status: Closed
This issue was found fixed in version 2.

11. Public function that could be declared external

Description

The following public functions that are never called by the contract should be declared external to save gas:

initialize()
setPredicate()
burnByAddress()

Remediation
Use the external attribute for functions never called from the contract.

Status: Closed
This issue was found fixed in version 2.

12. Avoiding Initial Values in Field Declarations

Description

The state variables mintFee and burnFee are declared outside the initialize() function. This is equivalent to setting these values in the constructor, and as such, will not work for upgradeable contracts.

Remediation

As recommended by the Openzappelin, make sure that all initial values are set in an initializer function.

Status: Closed

This issue was found fixed in version 2.

13. Typos in comments

Description

There are typos in the code comments.

L28: //dao users **mappin g**

L92: * @dev Set the dao users **accunt** id

L154: * @dev Creates `amount` tokens and **ssigns** them to `account`, increasing

Remediation

We recommend correcting the typos in the above comments, as follows:

L28: //dao users **mapping**

L92: * @dev Set the dao users **accoun**t id

L154: * @dev Creates `amount` tokens and **assigns** them to `account`, increasing

Status: Closed

This issue was found fixed in version 2.

Functional test

Function Names	Testing results
approve	Passed
burn	Passed
burnByAddress	Passed
burnByOwner	Passed
burnFrom	Passed
decreaseAllowance	Passed
increaseAllowance	Passed
initialize	Passed
mintByOwner	Passed
rounceOwnership	Passed
setBurnFee	Passed
setDAOUser	Passed
setDAOUserAccountID	Passed
setDAOUserFees	Passed
setData	Passed
setMintFee	Passed
setPredicate	Passed
transfer	Passed
transferFrom	Passed
transferOwnership	Passed

Automated Testing

Slither

```
INFO:Detectors:
OwnableUpgradeable.__gap (@openzeppelin/contracts-upgradeable/access/
OwnableUpgradeable.sol#74) shadows:
    - ContextUpgradeable.__gap (@openzeppelin/contracts-upgradeable/utils/
ContextUpgradeable.sol#31)
ERC20Upgradeable.__gap (@openzeppelin/contracts-upgradeable/token/ERC20/
ERC20Upgradeable.sol#310) shadows:
    - ContextUpgradeable.__gap (@openzeppelin/contracts-upgradeable/utils/
ContextUpgradeable.sol#31)
ERC20BurnableUpgradeable.__gap (@openzeppelin/contracts-upgradeable/token/ERC20/
extensions/ERC20BurnableUpgradeable.sol#48) shadows:
    - ERC20Upgradeable.__gap (@openzeppelin/contracts-upgradeable/token/ERC20/
ERC20Upgradeable.sol#310)
    - ContextUpgradeable.__gap (@openzeppelin/contracts-upgradeable/utils/
ContextUpgradeable.sol#31)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-
shadowing
INFO:Detectors:
BTCpx.initialize(address)._predicate (BTCpx.sol#42) lacks a zero-check on :
        - predicate = _predicate (BTCpx.sol#43)
BTCpx.setPredicate(address)._predicate (BTCpx.sol#49) lacks a zero-check on :
        - predicate = _predicate (BTCpx.sol#51)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-
address-validation
INFO:Detectors:
Different versions of Solidity is used:
    - Version used: ['^0.8.0', '^0.8.4']
    - ^0.8.0 (@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#3)
    - ^0.8.0 (@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
    - ^0.8.0 (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#3)
    - ^0.8.0 (@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#3)
    - ^0.8.0 (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/
ERC20BurnableUpgradeable.sol#3)
    - ^0.8.0 (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/
IERC20MetadataUpgradeable.sol#3)
    - ^0.8.0 (@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#3)
    - ^0.8.0 (@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#3)
    - ^0.8.4 (BTCpx.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-
directives-are-used
```

```
INFO:Detectors:
ContextUpgradeable.__Context_init() (@openzeppelin/contracts-upgradeable/utils/
ContextUpgradeable.sol#17-19) is never used and should be removed
ContextUpgradeable._msgData() (@openzeppelin/contracts-upgradeable/utils/
ContextUpgradeable.sol#27-30) is never used and should be removed
SafeMathUpgradeable.add(uint256,uint256) (@openzeppelin/contracts-upgradeable/utils/
math/SafeMathUpgradeable.sol#92-94) is never used and should be removed
SafeMathUpgradeable.div(uint256,uint256,string) (@openzeppelin/contracts-upgradeable/
utils/math/SafeMathUpgradeable.sol#190-195) is never used and should be removed
SafeMathUpgradeable.mod(uint256,uint256) (@openzeppelin/contracts-upgradeable/utils/
math/SafeMathUpgradeable.sol#150-152) is never used and should be removed
SafeMathUpgradeable.mod(uint256,uint256,string) (@openzeppelin/contracts-upgradeable/
utils/math/SafeMathUpgradeable.sol#212-217) is never used and should be removed
SafeMathUpgradeable.sub(uint256,uint256,string) (@openzeppelin/contracts-upgradeable/
utils/math/SafeMathUpgradeable.sol#167-172) is never used and should be removed
SafeMathUpgradeable.tryAdd(uint256,uint256) (@openzeppelin/contracts-upgradeable/utils/
math/SafeMathUpgradeable.sol#21-27) is never used and should be removed
SafeMathUpgradeable.tryDiv(uint256,uint256) (@openzeppelin/contracts-upgradeable/utils/
math/SafeMathUpgradeable.sol#63-68) is never used and should be removed
SafeMathUpgradeable.tryMod(uint256,uint256) (@openzeppelin/contracts-upgradeable/utils/
math/SafeMathUpgradeable.sol#75-80) is never used and should be removed
SafeMathUpgradeable.tryMul(uint256,uint256) (@openzeppelin/contracts-upgradeable/utils/
math/SafeMathUpgradeable.sol#46-56) is never used and should be removed
SafeMathUpgradeable.trySub(uint256,uint256) (@openzeppelin/contracts-upgradeable/utils/
math/SafeMathUpgradeable.sol#34-39) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (@openzeppelin/contracts-upgradeable/access/
OwnableUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider
deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts-upgradeable/token/ERC20/
ERC20Upgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying
with 0.6.12/0.7.6
Pragma version^0.8.0
(@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#3)
necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/
ERC20BurnableUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider
deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/
IERC20MetadataUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider
deploying with 0.6.12/0.7.6
Pragma version^0.8.0
```

(@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (BTCpx.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Function OwnableUpgradeable.__Ownable_init() (@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#27-30) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#32-36) is not in mixedCase
Variable OwnableUpgradeable.__gap (@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#74) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init(string,string) (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#53-56) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#58-61) is not in mixedCase
Variable ERC20Upgradeable.__gap (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#310) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init() (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#15-18) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init_unchained() (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#20-21) is not in mixedCase
Variable ERC20BurnableUpgradeable.__gap (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#48) is not in mixedCase
Function ContextUpgradeable.__Context_init() (@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#17-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#31) is not in mixedCase
Parameter BTCpx.initialize(address)._predicate (BTCpx.sol#42) is not in mixedCase
Parameter BTCpx.setPredicate(address)._predicate (BTCpx.sol#49) is not in mixedCase
Parameter BTCpx.setData(bytes)._relayData (BTCpx.sol#66) is not in mixedCase
Parameter BTCpx.setDAOUser(address,uint256,uint256)._addr (BTCpx.sol#76) is not in mixedCase
Parameter BTCpx.setDAOUser(address,uint256,uint256)._mintFee (BTCpx.sol#76) is not in mixedCase
Parameter BTCpx.setDAOUser(address,uint256,uint256)._burnFee (BTCpx.sol#76) is not in mixedCase

```
Parameter BTCpx.setDAOUserFees(address,uint256,uint256)._addr (BTCpx.sol#86) is not in
mixedCase
Parameter BTCpx.setDAOUserFees(address,uint256,uint256)._mintFee (BTCpx.sol#86) is not
in mixedCase
Parameter BTCpx.setDAOUserFees(address,uint256,uint256)._burnFee (BTCpx.sol#86) is not
in mixedCase
Parameter BTCpx.setDAOUserAccountId(address,string)._addr (BTCpx.sol#95) is not in
mixedCase
Parameter BTCpx.setDAOUserAccountId(address,string)._accountId (BTCpx.sol#95) is not in
mixedCase
Parameter BTCpx.burnByAddress(bytes,uint256)._btcAddr (BTCpx.sol#131) is not in
mixedCase
Parameter BTCpx.burnByAddress(bytes,uint256)._amount (BTCpx.sol#131) is not in mixedCase
Parameter BTCpx.mint(address,uint256,uint256,uint256)._addr (BTCpx.sol#146) is not in
mixedCase
Parameter BTCpx.mint(address,uint256,uint256,uint256)._amount (BTCpx.sol#146) is not in
mixedCase
Parameter BTCpx.mint(address,uint256,uint256,uint256)._txFee (BTCpx.sol#146) is not in
mixedCase
Parameter BTCpx.mint(address,uint256,uint256,uint256)._uuid (BTCpx.sol#146) is not in
mixedCase
Parameter BTCpx.isDAOUser(address)._addr (BTCpx.sol#170) is not in mixedCase
Parameter BTCpx.getUserMintFee(address)._addr (BTCpx.sol#178) is not in mixedCase
Parameter BTCpx.getUserBurnFee(address)._addr (BTCpx.sol#190) is not in mixedCase
Parameter BTCpx.getDAOUserAccountId(address)._addr (BTCpx.sol#202) is not in mixedCase
Constant BTCpx.percentageDivider (BTCpx.sol#17) is not in
UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-
solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (@openzeppelin/contracts-upgradeable/utils/
ContextUpgradeable.sol#28)" inContextUpgradeable (@openzeppelin/contracts-
upgradeable/utils/ContextUpgradeable.sol#16-32)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
OwnableUpgradeable.__gap (@openzeppelin/contracts-upgradeable/access/
OwnableUpgradeable.sol#74) is never used in BTCpx (BTCpx.sol#9-223)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-
variables
INFO:Detectors:
renounceOwnership() should be declared external:
    - OwnableUpgradeable.renounceOwnership() (@openzeppelin/contracts-upgradeable/
access/OwnableUpgradeable.sol#60-63)
transferOwnership(address) should be declared external:
    - OwnableUpgradeable.transferOwnership(address) (@openzeppelin/contracts-
```

upgradeable/access/OwnableUpgradeable.sol#69-73)
name() should be declared external:
    - ERC20Upgradeable.name() (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#66-68)
symbol() should be declared external:
    - ERC20Upgradeable.symbol() (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#74-76)
decimals() should be declared external:
    - BTCpx.decimals() (BTCpx.sol#58-60)
    - ERC20Upgradeable.decimals() (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#91-93)
totalSupply() should be declared external:
    - ERC20Upgradeable.totalSupply() (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#98-100)
balanceOf(address) should be declared external:
    - ERC20Upgradeable.balanceOf(address) (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#105-107)
transfer(address,uint256) should be declared external:
    - ERC20Upgradeable.transfer(address,uint256) (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#117-120)
approve(address,uint256) should be declared external:
    - ERC20Upgradeable.approve(address,uint256) (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#136-139)
transferFrom(address,address,uint256) should be declared external:
    - ERC20Upgradeable.transferFrom(address,address,uint256) (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#154-162)
increaseAllowance(address,uint256) should be declared external:
    - ERC20Upgradeable.increaseAllowance(address,uint256) (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#176-179)
decreaseAllowance(address,uint256) should be declared external:
    - ERC20Upgradeable.decreaseAllowance(address,uint256) (@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#195-201)
burn(uint256) should be declared external:
    - ERC20BurnableUpgradeable.burn(uint256) (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#27-29)
burnFrom(address,uint256) should be declared external:
    - ERC20BurnableUpgradeable.burnFrom(address,uint256) (@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#42-47)
initialize(address) should be declared external:
    - BTCpx.initialize(address) (BTCpx.sol#42-47)
setPredicate(address) should be declared external:
    - BTCpx.setPredicate(address) (BTCpx.sol#49-52)
burnByAddress(bytes,uint256) should be declared external:
    - BTCpx.burnByAddress(bytes,uint256) (BTCpx.sol#131-135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

Mythril

```
==== Potential denial-of-service if block gas limit is reached ====
SWC ID: 128
Severity: Low
Contract: 0x6ED82b3cD6f9836F9aC480b2E10b2B674C30826a
Function name: increaseAllowance(address,uint256)
PC address: 14502
Estimated Gas Usage: 1820 - 2675
Potential denial-of-service if block gas limit is reached.
A storage modification is executed in a loop. Be aware that the transaction may fail to execute
if the loop is unbounded and the necessary gas exceeds the block gas limit.
--------------------
Initial State:

Account: [ATTACKER], balance: 0x1, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x40000000020100, nonce:0, storage:{}

Transaction Sequence:

Caller: [SOMEGUY], function: approve(address,uint256), txdata:
0x095ea7b3000000000000000000000001010101010101010101010101010101010101010101
0101010101010101010101010101010101010101010101010101010101010101, value: 0x0
Caller: [ATTACKER], function: increaseAllowance(address,uint256), txdata:
0x39509351000000000000000000000001010101010101010101010101010101010101010101
0101010101010101010101010101010101010101010101010101010101010101, value: 0x0

==== Potential denial-of-service if block gas limit is reached ====
SWC ID: 128
Severity: Low
Contract: 0x6ED82b3cD6f9836F9aC480b2E10b2B674C30826a
Function name: burnFrom(address,uint256)
PC address: 14727
Estimated Gas Usage: 1918 - 2773
Potential denial-of-service if block gas limit is reached.
A storage modification is executed in a loop. Be aware that the transaction may fail to execute
if the loop is unbounded and the necessary gas exceeds the block gas limit.
--------------------
Initial State:

Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x40041, nonce:0, storage:{}
```

```
Transaction Sequence:

Caller: [SOMEGUY], function: approve(address,uint256), txdata:
0x095ea7b3000000000000000000000001010101010101010101010101010101010101010
1010101010101010101010101010101010101010101010101010101010101, value: 0x0
Caller: [CREATOR], function: burnFrom(address,uint256), txdata:
0x79cc679000000000000000000000000101010101010101010101010101010101010100000
0000000000000000000000000000000000000000000000000000000000000, value:
0x0
```

THEO

```
enderphan@enderphan contracts % theo --rpc-http http://172.0.0.1:8545
The account's private key (input hidden)
>
Contract to interact with
> 0x6ED82b3cD6f9836F9aC480b2E10b2B674C30826a
Scanning for exploits in contract: 0x6ED82b3cD6f9836F9aC480b2E10b2B674C30826a
Could not connect to RPC server. Make sure that your node is running and that RPC
parameters are set correctly.
No exploits found. You're going to need to load some exploits.

Tools available in the console:
- `exploits` is an array of loaded exploits found by Mythril or read from a file
- `w3` an initialized instance of web3py for the provided HTTP RPC endpoint
- `dump()` writing a json representation of an object to a local file

Check the readme for more info:
https://github.com/cleanunicorn/theo

Theo version v0.8.2
```

SOLIUM

```
BTCpx.sol
  11:4      warning    Line contains trailing whitespace                  no-trailing-whitespace
  36:4      warning    Line contains trailing whitespace                  no-trailing-whitespace
  72:1      warning    Line contains trailing whitespace                  no-trailing-whitespace
  82:1      warning    Line contains trailing whitespace                  no-trailing-whitespace
  91:1      warning    Line contains trailing whitespace                  no-trailing-whitespace
  99:1      warning    Line contains trailing whitespace                  no-trailing-whitespace
  108:1     warning    Line contains trailing whitespace                  no-trailing-whitespace
  146:4     error      "mint": Avoid assigning to function parameters.    security/no-assign-params
  152:4     warning    Line contains trailing whitespace                  no-trailing-whitespace
  153:1     warning    Line contains trailing whitespace                  no-trailing-whitespace
  163:1     warning    Line contains trailing whitespace                  no-trailing-whitespace
```

SOLHINT LINTER

```
Linter results:
contracts/BTCpx.sol:3:1: Error: Compiler version ^0.8.4 does not satisfy the r semver
requirement
contracts/BTCpx.sol:17:29: Error: Constant name must be in capitalized SNAKE_CASE
```

Results

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

Closing Summary

In this report, we have considered the security of the BTC Proxy platform. We performed our audit according to the procedure described above.

The audit showed several high, medium, low, and informational severity issues. In the end, a number of significant issues were fixed or acknowledged by the Auditee.

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

audits@quillhash.com