QuillAudits

Audit Report
March, 2023

For

PROXY

Table of Content

Executive Summary

Project Name	Kryptoin
Overview	The BTCpx.sol contract contains code with the functionality of swapping, minting and burning tokens across multiple chains.
Timeline	First Audit: 9 January, 2023 to 25th January, 2023 Extra Review: 2nd March, 2023 to 7th March, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyse Kryptoin codebase for quality, security, and correctness. *https://github.com/Proxy-Protocol/BTCpx-ERC20/blob/eb62aa06f01a83eee62aa944ee731637feec6700/contracts/BTCpx.sol* **Branch Name:** main **Commit Hash:** eb62aa06f01a83eee62aa944ee731637feec6700
Fixed In	https://github.com/Proxy-Protocol/BTCpx-ERC20/commit/6833a3fb9542e0391844472af2e3a4a98af5e9e2
Extra Review	https://github.com/Proxy-Protocol/BTCpx-ERC20/commit/b03813800b9ca3f79d8e11f6a83bd2df47ea19c7

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	1	0	1	5
Partially Resolved Issues	1	3	0	0
Resolved Issues	0	1	0	0

Types of Severities

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 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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

High Severity Issues

1. Privileged account risks

Description

The setAdminInitially function on #L116 has no access controls and if this contract was monitored at deployment, a frontrunner could call this function and set themselves as admin before the intended admin can. This would leave over 95% of critical functionality in the hands of an unintended admin. Also, the predicate account privileges could be transferred unintentionally to the wrong address.

Remediation

Include some form of access control here or set the admin directly in the constructor to avoid this risk. To mitigate account transfer issues, consider two-step ownership / privileged account transfers which would need the new account to accept the privileges granted as well as a time delay which could cancel the transfer if it happened unintentionally

Reference: *Ownable2Step*

Status

Partially Resolved

2. Centralization issues

Description

The contract is heavily dependent on an admin account as a signer for onlyAdmin function calls. A compromised admin account will open up numerous attack vectors which can lead to loss of funds, trust and users.

Remediation

It is highly recommended to carefully manage the admin and other privileged accounts in this scope. To reduce the impact of compromise, the following security measures can be implemented: Multisignature wallets (to require multiple accounts signing transactions before execution), or a Governance mechanism behind tokens.

Status

Acknowledged

Medium Severity Issues

3. Missing sanity checks

Description

The mint and burn fees for daoUsers (set in #L164) and regular users (set in #L181 and #L190) do not have any sanity checks. The admin could set the fees above 100% each and create an unexpected scenario.
Also, on #L458 there is no check for the daoTreasuryAddress. A malicious admin could set another address as the dao treasury address and siphon funds intended to be sent there.

Recommendation

Include input validation checks to have a range/limit of the expected input. Also verify the addresses passed in, at least for zero address checks and have such critical changes occur only when appropriately authorized. Authorization could be via a multi-sig wallet or using governance protocols to avoid centralization risks as stated above.

Status

Resolved

4. Consider using the SafeERC20 wrapper for token transfers

Description

Some ERC20 tokens have no return values in their transfer and transferFrom functions which could lead to return values which do not accurately describe changes to state. A token that failed to transfer could still return a true value at the end of its function call.

Recommendation

Use the SafeERC20 wrapper, specifically the safeTransfer and safeTransferFrom instead of the regular transfer and transferFrom in the ERC20 standard to properly deal with the return values from tokens that do not conform to the standard.

Status

Partially Resolved

5. Missing test cases

Description

The codebase lacks any form of unit test coverage. It is advisable to have unit tests with greater than 95% coverage of the codebase to reduce unexpected functionality.

Recommendation

Include unit tests with greater than 95% coverage of the codebase.

Auditor's Comment: Some tests have been added but with limited coverage of the contract scope. *https://github.com/Proxy-Protocol/BTCpx-ERC20/tree/audit/updates/test*

Status

Partially Resolved

6. Failing Arithmetic

Description

The contract is set to check for inputs greater/equal to 0 and less/equal to 100. Since the codebase deals with unsigned integers, Solidity would not accept any values less than 0 and that makes the check obsolete.

When combined with other arithmetic functions like getMintBySwapAmountAndFee() and getBurnToBridgeAmountAndFee(), it produces rounding errors with small amount of tokens where token fee amounts get rounded down to 0 when fee is too low or amount is too low.

Recommendation

- Have robust unit test cases to catch issues like these.
- Expand and specify the allowable limits to accept input which do not lead to rounding errors.

Status

Partially Resolved

Low Severity Issues

7. Variable shadowing

Description

The local variable _amount is shadowed in the function call where it is defined. It is passed in as a parameter to the function call, named as the return variable for the getMintBtcAmount call, passed as a parameter to the internal mint call as well as the Mint event.

Recommendation

It is best practice to have variable names well defined, consider renaming the variable to avoid this issue.

Status

Acknowledged

Informational Issues

8. Cheaper Arithmetic Operations

Description

The SafeMath library import in #L7 is unnecessary since Solidity versions >0.8.0 have inbuilt underflow and overflow checks for arithmetic operations, and are cheaper.

Recommendation

Consider removing the SafeMath library and refactoring the code to account for changes.

Status

Acknowledged

9. Misleading comments

Description

Multiple comments are misleading and should be updated to match the intended code flow. #L23 and #L25 can be set to any possible value and as such the comments would be invalid when the fees are updated, #L49 does not map hashes to address, #L225 assigns tokens to addr and not account, the token burn in #L196 is done by the admin not owner

Recommendation

Update the comments to match code functionality.

Status

Acknowledged

10. Address updates

Description

The setPredicate function does not check if predicate == _predicate. An event is emitted here everytime the function is called thus populating the blockchain with events that aren't true updates.

Recommendation

Consider adding a check for the above scenario.

Status

Acknowledged

11. Function visibility modifier

Description

The decimals getter function does not modify or read from state in any way because it returns a static uint8 value of 8. It does not need to have a view modifier.

Recommendation

Consider updating the view modifier to pure.

Status

Acknowledged

12. Unlocked pragma (#L3)

Description

The solidity pragma version in this codebase is unlocked.

Recommendation

It is always advisable to use a specific solidity version when deploying to production to reduce the surface of attack with future releases.

Status

Acknowledged

Recommendations

1. Consider indexing events for search ease and use of monitoring tools.

2. Gas optimizations

```
// original


 function tokenController(address _tokenAddress, bool _status) public onlyAdmin {
      require(tokens[_tokenAddress].status != _status, ...);
      Token memory token = tokens[_tokenAddress];
      token.status = _status;
      ...
  }


// reduced storage reading
 function tokenController(address _tokenAddress, bool _status) public onlyAdmin {
      Token memory token = tokens[_tokenAddress];
      require(token.status != _status, ...);
      token.status = _status;
      ...
 }

//original
 function mintBySwap(address _tokenAddress, uint256 _amount) public {
      ...
  IERC20Upgradeable token = IERC20Upgradeable(tokens[_tokenAddress].tokenAddress);



//unnecessary storage access
 function mintBySwap(address _tokenAddress, uint256 _amount) public {
      ...
  IERC20Upgradeable token = IERC20Upgradeable(_tokenAddress);
```

Automated Tests

```
BTCpx.feesCalculation(uint256,uint256) (contracts/BTCpx.sol#483-485) performs a multiplication on the result of a division:
        -(amount.div(PERCENTAGE_DIVIDER)).mul(_margin) (contracts/BTCpx.sol#484)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

BTCpx.addTokens(string,address,bool)._name (contracts/BTCpx.sol#348) shadows:
        - ERC20Upgradeable._name (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#43) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

BTCpx.setAdminInitially(address)._newAdmin (contracts/BTCpx.sol#119) lacks a zero-check on :
                - admin = _newAdmin (contracts/BTCpx.sol#122)
BTCpx.setDaoTreasury(address)._daoTreasuryAddress (contracts/BTCpx.sol#474) lacks a zero-check on :
                - daoTreasuryAddress = _daoTreasuryAddress (contracts/BTCpx.sol#475)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in BTCpx.mintBySwap(address,uint256) (contracts/BTCpx.sol#404-431):
        External calls:
        - success = token.transferFrom(msg.sender,address(this),_amount) (contracts/BTCpx.sol#415)
        State variables written after the call(s):
        - _mint(msg.sender,(_amount.sub(deduction))) (contracts/BTCpx.sol#422)
                - _balances[account] += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#272)
        - _mint(daoTreasuryAddress,deduction) (contracts/BTCpx.sol#423)
                - _balances[account] += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#272)
        - _mint(msg.sender,(_amount.sub(deduction_scope_0))) (contracts/BTCpx.sol#427)
                - _balances[account] += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#272)
        - _mint(daoTreasuryAddress,deduction_scope_0) (contracts/BTCpx.sol#428)
                - _balances[account] += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#272)
        - _mint(msg.sender,(_amount.sub(deduction))) (contracts/BTCpx.sol#422)
                - _totalSupply += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#269)
        - _mint(daoTreasuryAddress,deduction) (contracts/BTCpx.sol#423)
                - _totalSupply += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#269)
        - _mint(msg.sender,(_amount.sub(deduction_scope_0))) (contracts/BTCpx.sol#427)
                - _totalSupply += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#269)
        - _mint(daoTreasuryAddress,deduction_scope_0) (contracts/BTCpx.sol#428)
                - _totalSupply += amount (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#269)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in BTCpx.mintBySwap(address,uint256) (contracts/BTCpx.sol#404-431):
        External calls:
        - success = token.transferFrom(msg.sender,address(this),_amount) (contracts/BTCpx.sol#415)
        Event emitted after the call(s):
        - MintBySwap(address(this),msg.sender,(_amount.sub(deduction))) (contracts/BTCpx.sol#424)
        - MintBySwap(address(this),msg.sender,(_amount.sub(deduction_scope_0))) (contracts/BTCpx.sol#429)
        - Transfer(address(0),account,amount) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#274)
                - _mint(daoTreasuryAddress,deduction) (contracts/BTCpx.sol#423)
        - Transfer(address(0),account,amount) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#274)
                - _mint(msg.sender,(_amount.sub(deduction))) (contracts/BTCpx.sol#422)
        - Transfer(address(0),account,amount) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#274)
                - _mint(daoTreasuryAddress,deduction_scope_0) (contracts/BTCpx.sol#428)
        - Transfer(address(0),account,amount) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#274)
                - _mint(msg.sender,(_amount.sub(deduction_scope_0))) (contracts/BTCpx.sol#427)
Reentrancy in BTCpx.recoverToken(address,uint256) (contracts/BTCpx.sol#330-333):
        External calls:
        - IERC20Upgradeable(token).safeTransfer(admin,amount) (contracts/BTCpx.sol#331)
        Event emitted after the call(s):
        - RecoverToken(token,amount) (contracts/BTCpx.sol#332)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
AddressUpgradeable._revert(bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#206-218) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#211-214)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.2', '^0.8.4']
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#4)
        - ^0.8.4 (contracts/BTCpx.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/IERC20MetadataUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/math/SafeMathUpgradeable.sol#4) allows old versions
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#155-162):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Function ERC20Upgradeable.__ERC20_init(string,string) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#55-57) is not in mixedCase
Function ERC20Upgradeable.__ERC20_init_unchained(string,string) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#59-62) is not in mixedCase
Variable ERC20Upgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#400) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#16-17) is not in mixedCase
Function ERC20BurnableUpgradeable.__ERC20Burnable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#19-20) is not in mixedCase
Variable ERC20BurnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#51) is not in mixedCase
Function IERC20PermitUpgradeable.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/draft-IERC20PermitUpgradeable.sol#59) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Parameter BTCpx.initialize(address)._predicate (contracts/BTCpx.sol#94) is not in mixedCase
Parameter BTCpx.setAdmin(address)._newAdmin (contracts/BTCpx.sol#108) is not in mixedCase
Parameter BTCpx.setAdminInitially(address)._newAdmin (contracts/BTCpx.sol#119) is not in mixedCase
Parameter BTCpx.setPredicate(address)._predicate (contracts/BTCpx.sol#129) is not in mixedCase
Parameter BTCpx.setData(bytes)._relayData (contracts/BTCpx.sol#149) is not in mixedCase
Parameter BTCpx.setDAOUser(address,uint256,uint256)._addr (contracts/BTCpx.sol#159) is not in mixedCase
Parameter BTCpx.setDAOUser(address,uint256,uint256)._mintFee (contracts/BTCpx.sol#159) is not in mixedCase
```

```
Parameter BTCpx.setDAOUser(address,uint256,uint256)._burnFee (contracts/BTCpx.sol#159) is not in mixedCase
Parameter BTCpx.setDAOUserFees(address,uint256,uint256)._addr (contracts/BTCpx.sol#169) is not in mixedCase
Parameter BTCpx.setDAOUserFees(address,uint256,uint256)._mintFee (contracts/BTCpx.sol#169) is not in mixedCase
Parameter BTCpx.setDAOUserFees(address,uint256,uint256)._burnFee (contracts/BTCpx.sol#169) is not in mixedCase
Parameter BTCpx.setDAOUserAccountId(address,string)._addr (contracts/BTCpx.sol#178) is not in mixedCase
Parameter BTCpx.setDAOUserAccountId(address,string)._accountId (contracts/BTCpx.sol#178) is not in mixedCase
Parameter BTCpx.burnByAddress(bytes,uint256)._btcAddr (contracts/BTCpx.sol#228) is not in mixedCase
Parameter BTCpx.burnByAddress(bytes,uint256)._amount (contracts/BTCpx.sol#228) is not in mixedCase
Parameter BTCpx.mint(address,uint256,uint256,uint256)._addr (contracts/BTCpx.sol#244) is not in mixedCase
Parameter BTCpx.mint(address,uint256,uint256,uint256)._amount (contracts/BTCpx.sol#244) is not in mixedCase
Parameter BTCpx.mint(address,uint256,uint256,uint256)._txFee (contracts/BTCpx.sol#244) is not in mixedCase
Parameter BTCpx.mint(address,uint256,uint256,uint256)._uuid (contracts/BTCpx.sol#244) is not in mixedCase
Parameter BTCpx.isDAOUser(address)._addr (contracts/BTCpx.sol#269) is not in mixedCase
Parameter BTCpx.getUserMintFee(address)._addr (contracts/BTCpx.sol#279) is not in mixedCase
Parameter BTCpx.getUserBurnFee(address)._addr (contracts/BTCpx.sol#293) is not in mixedCase
Parameter BTCpx.getDAOUserAccountId(address)._addr (contracts/BTCpx.sol#305) is not in mixedCase
Parameter BTCpx.removeDAOUser(address)._addr (contracts/BTCpx.sol#339) is not in mixedCase
Parameter BTCpx.addTokens(string,address,bool)._name (contracts/BTCpx.sol#348) is not in mixedCase
Parameter BTCpx.addTokens(string,address,bool)._tokenAddress (contracts/BTCpx.sol#348) is not in mixedCase
Parameter BTCpx.addTokens(string,address,bool)._status (contracts/BTCpx.sol#348) is not in mixedCase
Parameter BTCpx.tokenController(address,bool)._tokenAddress (contracts/BTCpx.sol#360) is not in mixedCase
Parameter BTCpx.tokenController(address,bool)._status (contracts/BTCpx.sol#360) is not in mixedCase
Parameter BTCpx.viewStatus(address)._tokenAddress (contracts/BTCpx.sol#371) is not in mixedCase
Parameter BTCpx.setBridgeSigner(address)._account (contracts/BTCpx.sol#381) is not in mixedCase
Parameter BTCpx.chainController(string,bool)._chainName (contracts/BTCpx.sol#392) is not in mixedCase
Parameter BTCpx.chainController(string,bool)._status (contracts/BTCpx.sol#392) is not in mixedCase
Parameter BTCpx.mintBySwap(address,uint256)._tokenAddress (contracts/BTCpx.sol#404) is not in mixedCase
Parameter BTCpx.mintBySwap(address,uint256)._amount (contracts/BTCpx.sol#404) is not in mixedCase
Parameter BTCpx.burnToBridge(uint256,string)._amount (contracts/BTCpx.sol#438) is not in mixedCase
Parameter BTCpx.burnToBridge(uint256,string)._chainName (contracts/BTCpx.sol#438) is not in mixedCase
Parameter BTCpx.mintToBridge(address,uint256)._to (contracts/BTCpx.sol#464) is not in mixedCase
Parameter BTCpx.mintToBridge(address,uint256)._amount (contracts/BTCpx.sol#464) is not in mixedCase
Parameter BTCpx.setDaoTreasury(address)._daoTreasuryAddress (contracts/BTCpx.sol#474) is not in mixedCase
Parameter BTCpx.feesCalculation(uint256,uint256)._margin (contracts/BTCpx.sol#483) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

ERC20BurnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#51) is never used in BTCpx (contracts/BTCpx.sol#18-487)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

name() should be declared external:
        - ERC20Upgradeable.name() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#67-69)
symbol() should be declared external:
        - ERC20Upgradeable.symbol() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#75-77)
decimals() should be declared external:
        - BTCpx.decimals() (contracts/BTCpx.sol#140-142)
        - ERC20Upgradeable.decimals() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#92-94)
totalSupply() should be declared external:
        - ERC20Upgradeable.totalSupply() (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#99-101)
approve(address,uint256) should be declared external:
        - ERC20Upgradeable.approve(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#141-145)
transferFrom(address,address,uint256) should be declared external:
        - ERC20Upgradeable.transferFrom(address,address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#163-172)
increaseAllowance(address,uint256) should be declared external:
        - ERC20Upgradeable.increaseAllowance(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol#186-190)
decreaseAllowance(address,uint256) should be declared external:
```

```
burn(uint256) should be declared external:
        - ERC20BurnableUpgradeable.burn(uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#26-28)
burnFrom(address,uint256) should be declared external:
        - ERC20BurnableUpgradeable.burnFrom(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol#41-44)
addTokens(string,address,bool) should be declared external:
        - BTCpx.addTokens(string,address,bool) (contracts/BTCpx.sol#348-354)
tokenController(address,bool) should be declared external:
        - BTCpx.tokenController(address,bool) (contracts/BTCpx.sol#360-365)
viewStatus(address) should be declared external:
        - BTCpx.viewStatus(address) (contracts/BTCpx.sol#371-374)
setBridgeSigner(address) should be declared external:
        - BTCpx.setBridgeSigner(address) (contracts/BTCpx.sol#381-385)
chainController(string,bool) should be declared external:
        - BTCpx.chainController(string,bool) (contracts/BTCpx.sol#392-397)
mintBySwap(address,uint256) should be declared external:
        - BTCpx.mintBySwap(address,uint256) (contracts/BTCpx.sol#404-431)
burnToBridge(uint256,string) should be declared external:
        - BTCpx.burnToBridge(uint256,string) (contracts/BTCpx.sol#438-457)
mintToBridge(address,uint256) should be declared external:
        - BTCpx.mintToBridge(address,uint256) (contracts/BTCpx.sol#464-467)
setDaoTreasury(address) should be declared external:
        - BTCpx.setDaoTreasury(address) (contracts/BTCpx.sol#474-476)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

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 kryptoin codebase. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low, and Informational severity were found. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

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

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
March, 2023

For

PROXY

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com