Quantstamp Security Assessment Certificate

February 25th 2023 — Quantstamp Verified

TeleportDAO

This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Bridge
Auditors	Souhail Mssassi, Research Engineer Roman Rohleder, Research Engineer Nikita Belenkov, Security Auditor Ed Zulkoski, Senior Security Engineer Martinet Lee, Senior Research Engineer
Timeline	2022-10-05 through 2022-12-03
EVM	Arrow Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Documentation
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
TeleportDAO/bitcoin-evm-smart-contracts	7871cad initial audit

Total Issues	**42** (31 Resolved)
High Risk Issues	**11** (10 Resolved)
Medium Risk Issues	**6** (4 Resolved)
Low Risk Issues	**13** (7 Resolved)
Informational Risk Issues	**9** (8 Resolved)
Undetermined Risk Issues	**3** (2 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Initial audit:
Through reviewing the code, we found **23 potential issues** of various levels of severity: 1 high-severity, 3 medium-severity, 7 low-severity, 9 informational-severity, 3 undetermined-severity. The high severity issues represent a real risk for TeleportDAO. We recommend addressing all the issues before deploying the code. To note also that a very sensitive part of the protocol, the parsing of bitcoin transaction data, is done by modified third-party libraries (https://github.com/summa-tx/memview-sol and https://github.com/summa-tx/bitcoin-spv/tree/master/solidity), which themselves are highly complex but come with little to no tests, we recommend adding more tests.
Additional audit: Quantstamp performed additional security review with two auditors and found 13 additional issues: 5 high-severity, 2 medium-severity, and 6 low-severity. The issues ranges from protocol level design to implementation: e.g. malicious locker can steal funds in certain situations, user funds can be locked, and exchanges were used as oracle directly which are subject to flashloan attacks. While most issues are fixed, some are acknowledged or mitigated only on the UI level, protocol level design is a delicate matter and we believe it warrants further attention.
Additional findings: The team discovered further issues post audit and Quantstamp verified the issue and the fixes. For the sake of completeness, it is being included in the report. During the verification process, Quantstamp also discovered an issue of the updated code "Possible to Buy Collateral without Burning Telebtc" and the team fixed it subsequently. The latest fix were verified on: `97f2527`

ID	Description	Severity	Status
QSP-1	Test Code Mixed with Production Code	High	Fixed
QSP-2	Instant Router Susceptible to Flash Loan Attacks	High	Fixed
QSP-3	`PriceOracle.equivalentOutputAmount()` Is Using Data From Exchange when Chainlink Is Not Available for the Given Asset.	High	Fixed
QSP-4	Lockers May Steal Funds Away From Users Before `teleBTC` Is Minted	High	Fixed
QSP-5	User Funds Could Be Locked Due to Locker Capacity Being Reached	High	Acknowledged
QSP-6	Lockers Can Steal Funds but Still Avoid Slashing by Supplying Repeated Index	High	Fixed
QSP-7	Infinite Slashing in Disbutelocker	High	Fixed
QSP-8	Locker Can Avoid Slashing by Forcing the Slashing Calculation to Revert	High	Fixed
QSP-9	Draining the Collateral Gradually Using Precision Loss	High	Fixed
QSP-10	User Can Make Unfulfillable Request to Slash Lockers	High	Fixed
QSP-11	Possible to Buy Collateral without Burning Telebtc	High	Fixed
QSP-12	Locker Can Be Slashed Again if a New `ccBurnRouter`contract Is Being Deployed and Used	Medium	Fixed
QSP-13	Safetransfer Should Be Used	Medium	Fixed
QSP-14	Inaccurate Accounting in the Collateral Pool if Deflationary Tokens Are Accepted	Medium	Acknowledged
QSP-15	Anyone Can Call `payBackLoan()`	Medium	Acknowledged
QSP-16	An Honest Locker Can Be Slashed	Medium	Fixed
QSP-17	Btcrelay Cannot Recover if Reorg Happens	Medium	Fixed
QSP-18	ccBurnRouter Is Not Set In An Initializer	Low	Fixed
QSP-19	Front Running Potential Between `burnProof()` and `disputeBurn()`	Low	Acknowledged
QSP-20	Missing Input Validation	Low	Mitigated
QSP-21	TeleBTC Doesn't Get Set In The Constructor	Low	Fixed
QSP-22	Renounce Ownership of Certain Contracts	Low	Fixed
QSP-23	Privileged Roles and Ownership	Low	Acknowledged
QSP-24	Potentially Faulty Use of `Uniswap` Interfaces	Low	Fixed
QSP-25	Highly Utilized Pools May Be Difficult to Exit	Low	Acknowledged
QSP-26	Cannot Slash if Exchange Is Illiquid	Low	Acknowledged
QSP-27	Configuration Requirements Not Strictly Enforced	Low	Fixed
QSP-28	Relayer May Not Be Compensated	Low	Acknowledged
QSP-29	Estimations of Fee May Change Unintentionally by Changing `epochLength`	Low	Acknowledged
QSP-30	Unnecessary Precision Loss Due to Suboptimal Arithmetic Order	Low	Fixed
QSP-31	Block Timestamp Manipulation	Informational	Acknowledged
QSP-32	Unlocked Pragma	Informational	Fixed
QSP-33	Application Monitoring Can Be Improved by Emitting More Events	Informational	Mitigated
QSP-34	Checks-Effects-Interactions Pattern Violations	Informational	Mitigated
QSP-35	Timestamp Potentially Overflowable in `PriceOracle.equivalentOutputAmount()`	Informational	Fixed
QSP-36	Clone-and-Own	Informational	Fixed
QSP-37	Remove-Loops Gas-Optimizable	Informational	Fixed
QSP-38	Commented-Out Code	Informational	Mitigated
QSP-39	Unnecessary Use of `SafeMath` in Solidity 0.8.x	Informational	Fixed
QSP-40	No Incentive for Teleporter to Slash User when Locked Collateral Is Lower than the Loan Amount	Undetermined	Acknowledged
QSP-41	`TypedMemView.encodeHex()` Always Reverts	Undetermined	Fixed
QSP-42	Length Overflowable in `BitcoinHelper.revertNonMinimal()`	Undetermined	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

DISCLAIMER:
If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Test Code Mixed with Production Code

Severity: *High Risk*

Status: Fixed

File(s) affected: `erc20/ERC20.sol`, `erc20/TeleBTC.sol`

Description: To prevent confusion and potentially introduce vulnerabilities, test code should not be present in production code/should be clearly separated.
The following instances have been noted:

1. Several contracts import a test-related contract (`import "hardhat/console.sol";`).
2. `TeleBTC.mintTestToken()`: Allows an arbitrary address to mint 100 TeleBTC to oneself (`10000000000`, with `decimals` being `8`) an unlimited number of times.

Recommendation: Remove any test-related functionality from production code.

Update: The team has fixed by removing `TeleBTC.mintTestToken()` and all occurrences of the import of `hardhat/console.sol`, as suggested.

QSP-2 Instant Router Susceptible to Flash Loan Attacks

Severity: *High Risk*

Status: Fixed

File(s) affected: `InstantRouter.sol`

Description: As the `InstantRouter` effectively uses the Uniswap exchange as an oracle, it may be susceptible to flash loan attacks.

Exploit Scenario: 1. Suppose `100 collateralToken == 1 teleBTC`, and the Uniswap exchange is on par with this amount. Let's also assume the collateral ratio is 150%.

1. Take out instant loan for 1 teleBTC, locking 150 collateralToken.
2. Wait out the loan period.
3. In a single transaction: a. Borrow a huge amount of teleBTC and deposit into the exchange, manipulating the exchange rate b. Invoke `InstantRouter.slashUser()` on themself: - The function will use the exchange to try to convert the 150 collateralToken back into 1 teleBTC. - Since the exchange rate is manipulated, this may require way less than 150 collateralToken to make them solvent. Suppose it only required exchanging 50 collateralToken, and therefore the user still has an excess of 100 collateralTokens after the slash. - Suppose the `slasherPercentageReward` is 20%. Of the remaining 100 collateralTokens, 80 will now go back into the user's account, and 20 will go to the slasher (important - these are the same person). c. Unroll the flash loan.
4. Net result: The user has 1 teleBTC. The user has 100 collateralToken (minus fees incurred by Uniswap+flash loans). So they've effectively paid 50 collateralToken for 1 teleBTC (basically a 50% discount).

Recommendation: Avoid using the Uniswap exchange as the sole oracle. **Note:** this issue may also allow users to manipulate `slashUser()` calls.

Update: The code has been updated to attain price from oracles and compare them to the Uniswap price and only accepts if there are 10% of difference. Any extra collateral are sent to the treasury to demotivate potential malicious user thus fixing the issue.

QSP-3 `PriceOracle.equivalentOutputAmount()` Is Using Data From Exchange when Chainlink Is Not Available for the Given Asset.

Severity: *High Risk*

Status: Fixed

File(s) affected: `LockersLib.sol`, `InstantRouter.sol`, `PriceOracle.sol`

Description: When Chainlink pricefeed is not available or out of date, the function `equivalentOutputAmount()` uses the price from exchange routers (e.g. Uniswap) and take an average on all the data sources. Note that these sources can be manipulated with flashloan and skew the result.
The funciton is used in:

- `LockersLib.slashThiefLocker()` (which is used by `CCBurnRouter._slashLockerForDispute()`) (teleBTC to NativeToken)
- `LockersLib.slashIdleLocker()` (used by `CCBurnRouter.disputeBurn()`) (NativeToken to teleBTC)
- `LockersLib.priceOfOneUnitOfCollateralInBTC()` (NativeToken to teleBTC)
- `LockersLib.lockerCollateralInTeleBTC()` which is used by `LockersLib.getLockerCapacity()` (which is in turn used by `LockersLib.mint()`) (NativeToken to teleBTC)
- `InstantRouter._lockCollateral()`. These are indirectly used by `InstantRouter.instantCCTransfer()` and `InstantRouter.instantCCExchange()`.
  . collateral tokens to teleBTC. Collateral tokens are whitelisted in `collateralPoolFactory`.

From our PoV, adding the fallback to average with price of instant liquidity pools introduces unnecessary operational risk. The system would not work if chainlink or other reliable and non-manipulable oracles are used.
The potential impacts (not limited to) when chainlink is not configured are:

- ineffective slashing
- collateral not properly locked
- minting more teleBTC than capacity
- slashing healthy lockers or users

Recommendation: We recommend not to use external liquidity pools as a fallback when querying `equivalentOutputAmount()`. If the fallback is added for view purposes, we recommend to add a function that is not manipulable and use the non-manipulable version in the smart contract code.

Update: The issue was fixed by only using the oracle data. However, the timestamp of oracle return value has been disregarded within the implementation. While this issue is fixed, it makes the system vulnerable if the oracle data becomes stale.
Further Update: The team has added check for the oracle timestamp. The acceptable timestamp delay is controlled by the owner through `acceptableDelay`. Additionally, the team says that the function `equivalentOutputAmountByAverage()`(which sources some data from exchanges) should only be used in UI/Front-end and not in smart contracts.

QSP-4 Lockers May Steal Funds Away From Users Before `teleBTC` Is Minted

Severity: *High Risk*

Status: Fixed

Description: Users can only call "disputeLocker" when the `_inputTxId` is confirmed on the relayer. The `input tx` according to the comments is the malicious tx that shows that the locker moved the BTC.

Exploit Scenario: 1. User send BTC to Locker. 2. Before the send BTC tx is finalized in the BTCRelayer contract (3 blocks), users cannot mint teleBTC. Hence this is the time when the locker still has all his capacity and can remove all the collaterals on ETH. Once the transaction that sends the BTC to the locker is mined in at least one block, the malicious relayer removes collateral and transfers the BTC away. 3. The users now need to race with the tx he sent earlier. I believe there is a higher probablity that the users will lose since it is already mined.

Recommendation: Consider an alternative design or clarify the mitigation of the issue. A potential mitigation strategy is to add a delay in `removeCollateral()`.

Update: The team added a delay to remove collateral. A locker can only remove collateral after the delay is passed and the delay should be configured to be higher than the finalization

parameter. This removes the possibility of locker removing collateral and running away with the BTC in their address.

QSP-5 User Funds Could Be Locked Due to Locker Capacity Being Reached

Severity: *High Risk*

Status: Acknowledged

Description: Multiple users can send funds to the locker while it still has capacity for individual users but not all of them combined. A variation of this, is the locker removes collateral and hence decreasing its capacity at the same time when a user sends fund to the locker.
In both cases, the teleporter cannot invoke `ccTransfer()` in `CCTransferRouter` since it exceeds the capacity of the locker. Since there are no requirement for the locker to send back the funds, the funds could be locked indefinitely. If the locker is malicious, then it also gives it more time to steal funds following the exploit steps in "Lockers may steal funds away from users before `teleBTC` is minted".

Recommendation: Consider an alternative design or clarify the mitigation of the issue.

Update: The team decided to mitigate this on the UI level. "We will prevent users to send requests in UI if the locker's capacity is low. If the locker's remaining capacity is X, we only accept requests in UI) that move at most X/10 from Bitcoin to the target chain. Also, to calculate the locker's remaining capacity, we will consider the pending requests (requests that have not been finalized yet but affect the locker's capacity in near future)". We believe the UI mitigation described is not sufficient as this should fail when the demand is high at the same time, hence we consider this as acknowledged.

QSP-6 Lockers Can Steal Funds but Still Avoid Slashing by Supplying Repeated Index

Severity: *High Risk*

Status: Fixed

Description: When lockers supply their burn proof, they submit the indices of `_vout` that would be used to match the burn requests. The indices array `_voutIndexes` were never checked for uniqueness and hence one could supply repeated indices. If there are multiple burn requests at the same size, a malicious locker can then have an output that matches its size, and use its index to nullify all the burn requests.

Exploit Scenario: 1. A user burnt through `ccBurn` 10 times with the same `_amount`. Hence the user should be getting `10 * _amount`

1. The locker sends a transaction with 10 output. One of which matches the `_amount`, but all others are `0`. For the sake of simplicity, the one that matches `_amount` is the first element (element `0`) in the `vout` array. The locker only sent `_amount` in total.
2. The locker supplies `burnProof` with the `vout` array and the `indices` array. However, the `indices` array is supplied with `[0,0,0, ..., 0]`.
3. Since the burnProof is submitted, from the system's point of view, the locker has fulfilled its promise and hence cannot be slashed.

Recommendation: Check the uniqueness of the index array.

Update: The team applied the recommendation and fixed the issue.

QSP-7 Infinite Slashing in Disbutelocker

Severity: *High Risk*

Status: Fixed

File(s) affected: `ccBurnRouter.sol`

Description: In `ccBurnRouter.disputeLocker()`, the user can provide proof of slashing infinite times to slash the locker.

Update: The team has fixed the vulnerability by setting `isUsedAsBurnProof[_inputTxId]`to true after the proof is being provided.

QSP-8 Locker Can Avoid Slashing by Forcing the Slashing Calculation to Revert

Severity: *High Risk*

Status: Fixed

File(s) affected: `LockersLib.sol`

Description: In `LockersLib.slashThiefLocker()`. The locker can transfer some BTC to itself, then move the total locker's BTC to another address. Here we should slash the locker, but since the BTC amount is greater than the locker's net minted TeleBTC, the contract will be reverted.

Update: The team fixed this by detecting the case and making the locker's net minted to zero.

QSP-9 Draining the Collateral Gradually Using Precision Loss

Severity: *High Risk*

Status: Fixed

File(s) affected: `LockersLib`

Description: in `LockersLib.buySlashedCollateralOfLocker()`, the user can drain the collateral by repeatedly buying a small amount of collateral. Due to precision loss, the required BTC is `0` and allows users to drain the collateral. Similar error is present in `LockersLib.liquidateLocker()`.

Update: The team has fixed it via adding `1` to the required btc. If the user wants to buy all remaining collateral, then the required btc would be the exact number that it was stored.

QSP-10 User Can Make Unfulfillable Request to Slash Lockers

Severity: *High Risk*

Status: Fixed

File(s) affected: `ccBurnRouter.sol`

Description: The user can submit a burn request that are too small for the locker to fulfill, hence making the locker slashable. Users can slash all of a locker's collateral by this way.

Update: The team fixed the issue by requesting a higher minimum amount when submitting a burn request.

QSP-11 Possible to Buy Collateral without Burning Telebtc

Severity: *High Risk*

Status: Fixed

File(s) affected: `LockersLib.sol`, `LockersLogic.sol`

Description: When a locker is being slashed, the system records `slashingTeleBTCAmount` and `reservedNativeTokenForSlash` at the same time. Users can later perform `buySlashedCollateralOfLocker()` to burn teleBTC and get the native token with a discount. However the logic does not consider the possible price fluctuation between `SlashTheifLocker()` and `buySlashedCollateralOfLocker()`, hence under certain condition, users can nearly burn minimal teleBTC to get asset.

Recommendation: Consider not having double accounting as it is the source of error.

Update: The team mitigated the double accounting's effect by always using the ratio between the recorded values and not using the market price between the two assets.

QSP-12 Locker Can Be Slashed Again if a New `ccBurnRouter`contract Is Being Deployed and Used

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `ccBurnRouter`

Description: As the slashing state were stored in the `ccBurnRouter`, if a new `ccBurnRouter` is being deployed and connected to the system, all the slashable events can be submitted again and locker can be slashed multiple times for the same thing.

Update: The team fixed it by introducing a variable `startingBlockNumber`to the contract to prevent transaction older than the `startingBlockNumber` will not be accepted.

QSP-13 Safetransfer Should Be Used

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `LockerLogic.sol`, `CCBurnRouter.sol`

Description: Since some ERC20 tokens do not return a value on below mentioned functions/behave non-uniformly it is advised to use a secure wrapping interface around said functions, like for example OpenZeppelins SafeERC20. When transferring ERC20s, it is good practice to use `safeTransfer(...)` and `safeTransferFrom(...)`, as it reverts if the transfer does not succeed. This should be added in the following cases:

1. `LockerLogic.revokeRequest()`
2. `LockerLogic._removeLocker()`
3. `CCBurnRouter.ccBurn()`
4. `BitcoinRelay._sendReward()`
5. `CollateralPool.addCollateral()`
6. `CollateralPool.removeCollateral()`
7. `InstantRouter._lockCollateral()`
8. `InstantRouter.paybackLoan()`
9. `InstantRouter.slashUser()`
10. `InstantPool.addLiquidity()`

This is especially important in `_lockCollateral()` in `InstantRouter.sol` since the transfer could fail, but the function would still mint the collateral equivalent token, and then the user can do an uncollateralized instant transfer.

Recommendation: Add `safeTransfer(...)` and `safeTransferFrom(...)` from OZ as suggested above.

Update: Mitigated by acknowledging cases involving the `TeleBTC` token (as it's behaviour is known and `safeTransfer` not needed) and fixing all others.

1. Fixed.
2. Fixed.
3. Acknowledged.
4. Fixed.
5. Fixed.
6. Fixed.
7. Fixed.
8. Fixed (TeleBTC transfers acknowledged).
9. Fixed.
10. Acknowledged.

QSP-14 Inaccurate Accounting in the Collateral Pool if Deflationary Tokens Are Accepted

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `pools/CollateralPool.sol`

Description: The `addCollateral()` function mints the equivalent of a deposit in tokens to be used for instant functionality. If deflationary tokens are accepted as collateral, the number of minted tokens will be inaccurate for collateral that subtracts fees during transfer. This could lead to undercollaterlized instant transfers.
The pools need to account for such tokens if they are planned to be used.

Recommendation: Consider whether deflationary tokens will be accepted for deposits. If yes, then account for the difference between the pre-transfer balance and the post-transfer balance of the receiving contract. The difference will capture the number of tokens that have been transferred to the receiving contract, regardless of the token transfer mechanism.

Update: The client has acknowledged the issue and commented on the following: "There is a selective list of tokens accepted as collateral. None of them are deflationary. And because the payback time is short and loans are over-collateralized, there will be no issue."

QSP-15 Anyone Can Call `payBackLoan()`

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `routers/InstantRouter.sol`

Description: Currently, anyone can call `payBackLoan()` function. Due to the logic structure and the interactions between different parts of the system, only `CCTransferRouter` should be able to call the `payBackLoan()` function.

Recommendation: A check of `msg.sender` should be added that verifies that the transaction's sender is `CCTransferRouter`.

Update: The client has acknowledged the issue and commented on the following: "In our protocol, anyone can pay back the loan. So it shouldn't necessarily be called by the CCTransferRouter contract. In particular, a user can call payBackLoan() and directly pay the loan back using wrapped tokens they own. This only opens more convenient options for paying back the loan for users and introduces no threats to them."

QSP-16 An Honest Locker Can Be Slashed

Severity: *Medium Risk*

Status: Fixed

Description: `burnProof()` requires the outward tx from the locker to be confirmed (block is finalized). However, the slashing in `disputeBurn()` is comparing the `_lastSubmittedHeight` with the deadline. Lockers can be slashed if `finalizationParameter` is being set to an unreasonable number (compared to `transferDeadline`).

Exploit Scenario: 1. a user burns its teleBTC. The deadline is `request.deadline = _lastSubmittedHeight + transferDeadline;` and let's suppose the ideal case: that the locker sees the event and immediately sends the BTC.

1. set `finalizationParameter` to be greater than `transferDeadline`.
2. After `transferDeadline` blocks has passed, the locker can be slashed already by `disputeBurn()`. However, there is still no way for the locker to perform `burnProof()` as `_isConfirmed()` requires the block that the locker sent the BTC to user to be finalized.

Recommendation: Consider also checking the `transferDeadline` while setting the `finalizationParameter`.

Update: The team added a function where everyone could call to set the `trasnferDeadline` to be `finalizationParameter*2 + 1`, thus fixing the issue.

QSP-17 Btcrelay Cannot Recover if Reorg Happens

Severity: *Medium Risk*

Status: Fixed

Description: The light client implementation here doesn't have any means to recover once they have finalized. While Bitcoin Re-org doesn't seem to happen too often, there is currently no recovery mechnism once the re-org affects the block that was marked finalized in the system.

Recommendation: Evaluate the risks of re-org. Given that the system is already heavily dependent on owner being benign, consider introduce a recovery mechanism that could be invoked by the owner.

Update: This is a false positive, while not through direct method, the owner can in fact recover the system. According to the team, "The owner can pause the contract and still add block headers. If the owner sees that a re-org happened, he will pause the contract, increase the finalization parameter and submit the new canonical chain."

QSP-18 ccBurnRouter Is Not Set In An Initializer

Severity: *Low Risk*

Status: Fixed

File(s) affected: `lockers/LockersLogic.sol`

Description: When the contract is initialized, parameters are passed to set the expected behaviour of the contract.
One of the interactions that the `LockersLogic` has is the ability to burn CC tokens via `ccBurnRouter`. Currently, `ccBurnRouter` is not passed as a parameter and is assigned the default address of 0x0. Unless the specific setter is called straight after the initialization, the contract will call methods on the 0x0 address and fail, hence bricking some contract functions.

Recommendation: Add `ccBurnRouter` as one of the initialization parameters.

Update: The team has fixed by adding `ccBurnRouter` as a parameter and setting it in the initializer, as suggested.

QSP-19 Front Running Potential Between `burnProof()` and `disputeBurn()`

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `routers/CCBurnRouter.sol`

Description: Due to the way `burnProof()` and `disputeBurn()` operate, there is a possibility that even when the locker behaves correctly, it can get slashed due to the ordering of the transactions.
If `disputeBurn` is executed before `burnProof`, the locker will get slashed even if it is behaving correctly.

Recommendation: Take this into account in the protocol design.

Update: The client has acknowledged the issue and commented on the following: "disputeBurn() can only be successfully executed when the locker has not acted honestly and has not provided the users with a proof of a payment to them in a timely manner. The lockers should pay the users before the deadline and provide the proof of that payment. If they provide the proof before the deadline as well, there is no way they get punished. If they hesitate to provide the proof they might get punished. If no one calls disputeBurn() after the deadline, they still have time to provide the proof but they are accepting the risk of getting punished at any moment."

QSP-20 Missing Input Validation

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `relay/BitcoinRelay.sol`, `routers/CCBurnRouter.sol`, `lockers/LockersLogic.sol`, `pools/CollateralPool.sol`, `oracle/PriceOracle.sol`

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following functions do not have a proper validation of input parameters:

1. `BitcoinRelay.constructor()` does not check that the `.hash256()` of parameter `_genesisHeader` is different from parameter `_periodStart`.
2. `BitcoinRelay.setRewardAmountInTDT()` does not check that parameter `_rewardAmountInTDT` is different from zero (or otherwise bound).
3. `BitcoinRelay.setFinalizationParameter()` does not check that parameter `_finalizationParameter` is different from zero (or otherwise bound).
4. `BitcoinRelay.setRelayerPercentageFee()` does not check that parameter `_relayerPercentageFee` is smaller than `100` (as stated in `BitcoinRelay.sol#L25`: `A number between [0, 100)`).
5. `BitcoinRelay.setEpochLength()` does not check that parameter `_epochLength` is different from zero (or otherwise bound).
6. `BitcoinRelay.setBaseQueries()` does not check that parameter `_baseQueries` is different from zero (or otherwise bound).
7. `BitcoinRelay.setSubmissionGasUsed()` does not check that parameter `_submissionGasUsed` is different from zero (or otherwise bound).
8. `CCBurnRouter.setBitcoinFee()` does not check that parameter `_bitcoinFee` is different from zero (or otherwise bound).
9. `LockersLogic.initialize()` does not check that parameter `_lockerPercentageFee` is non-zero and smaller than `MAX_LOCKER_FEE` (`10000`).
10. `CollateralPool.constructor()` does not check that parameter `_collateralizationRatio` is equal or greater than `100` (i.e. allows for under-collateralization).
11. `CollateralPool.setCollateralizationRatio()` does not check that parameter `_collateralizationRatio` is equal or greater than `100` (i.e. allows for under-collateralization).
12. `CollateralPoolFactory.createCollateralPool()` does not check that parameter `_collateralizationRatio` is equal or greater than `100` (i.e. allows for under-collateralization).
13. `InstantPool.constructor()` does not check that parameter `_instantPercentageFee` is equal to or less than `MAX_INSTANT_PERCENTAGE_FEE`.
14. `InstantPool.setInstantPercentageFee()` does not check that parameter `_instantPercentageFee` is equal to or less than `MAX_INSTANT_PERCENTAGE_FEE`.
15. `PriceOracle.setAcceptableDelay()` does not check that parameter `_acceptableDelay` is different from zero (or otherwise bound).

Recommendation: Adopt the suggested condition checks.

Update: 1. Acknowledged.

1. Acknowledged.
2. Fixed.
3. Fixed.
4. Fixed.
5. Fixed.
6. **Unresolved**.
7. Fixed.
8. **Unresolved**.
9. Fixed.
10. Fixed.
11. Fixed.
12. Fixed.
13. Fixed.
14. Fixed.

QSP-21 TeleBTC Doesn't Get Set In The Constructor

Severity: *Low Risk*

Status: Fixed

File(s) affected: `routers/CCBurnRouter.sol`

Description: When the contract is initialized, parameters are passed in to set the expected behaviour of the contract. In the case of the `CCBurnRouter`, the address of the TeleBTC contract is not set via the constructor and needs to be set via an additional function. This could lead to issues if the contract is not set correctly.

Recommendation: Add `TeleBTC` as one of the initialization parameters.

Update: the team has fixed the issue by adding `teleBTC` as a parameter and setting it in the constructor, as suggested

QSP-22 Renounce Ownership of Certain Contracts

Severity: *Low Risk*

Status: Fixed

File(s) affected: `lockers/LockerLogic.sol`, `oracle/PriceOracle.sol`

Description: OpenZeppelin's Ownable contract contains `renounceOwnership()` function which allows the owner to renounce ownership. If the owner calls `renounceOwnership()`, the contract would be left without an owner, making some aspects of the contract impossible to use.

Recommendation: Consider if ownership revocation is a necessary feature. If it is not, override the ownership revocation functionality to disable it. If it is, add end-user documentation stating the risk.

Update: the team has fixed the issue by overriding `renounceOwnership()` in all corresponding contracts with an empty function, as suggested.

QSP-23 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `erc20/TeleBTC.sol`, `/relay/BitcoinRelay.sol`, `routers/CCBurnRouter.sol`, `lockers/LockersLogic.sol`, `pools/CollateralPool.sol`, `pools/InstantPool.sol`, `oracle/PriceOracle.sol`, `routers/InstantRouter.sol`, `/contracts/routers/CCExchangeRouter.sol`, `routers/CCTransferRouter.sol`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.
The `TeleBTC.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
  . Assign a new `_owner` address by calling `transferOwnership()`.
  . Rennounce the role (**and thereby preventing any future calls to the following listed functions!**) by calling `renounceOwnership()`.
  . Add/Remove arbitrary addresses from the `minters[]` role by calling `addMinter()`/`removeMinter()`.
  . Add/Remove arbitrary addresses from the `burners[]` role by calling `addBurner()`/`removeBurner()`.
- `minters[]`, as set through `addMinter()` by `_owner`:
  . Mint an arbitrary amount of new tokens to an arbitrary address by calling `mint()`.
- `burners[]`, as set through `addBurner()` by `_owner`:
  . Burn an arbitrary amount of tokens they themselves are currently holding by calling `burn()`.

The `BitcoinRelay.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
  . Assign a new `_owner` address by calling `transferOwnership()`.
  . Renounce the role (**and thereby preventing any future calls to the following listed functions!**) by calling `renounceOwnership()`.
  . Set/Unset the `paused` state for the contract (and thereby control the ability of calling `checkTxProof()`,`addHeaders()` and `addHeadersWithRetarget()`) by calling `pauseRelay()`/`unpauseRelay()`.
  . Arbitrarily set the TeleportDAO token reward for relayers (**can be any arbitrary value, including zero!**) by calling `setRewardAmountInTDT()`.
  . Change the block finalization length by calling `setFinalizationParameter()` (at deployment time: `3`).
  . Arbitrarily set the native token percentage reward for relayers (**can be any arbitrariy value, including zero!**) by calling `setRelayerPercentageFee()` (at deployment time: `5`).
  . Arbitrarily set the fee-related block epoch length by calling `setEpochLength()` (at deployment time: `2016`).
  . Arbitrarily change the base query count by calling `setBaseQueries()`.
  . Arbitrarily change the submission gas amount (and thereby all corresponding fees) by calling `setSubmissionGasUsed()` (at deployment time: `300000`).
  . Add new headers to the chain, even when contract is in paused state by calling `ownerAddHeaders()`.
  . Add new headers (with retarget), even when contract is in paused state by calling `ownerAddHeadersWithRetarget()`.

The `CCBurnRouter.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
  . Assign a new `_owner` address by calling `transferOwnership()`.
  . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.
  . Change the relay contract address by calling `setRelay()`.
  . Change the locker contract address by calling `setLockers()`.
  . Change the TeleBTC token contract address by calling `setTeleBTC()`.
  . Change the treasury address by calling `setTreasury()`.
  . Change the transfer deadline parameter (being at least higher than `IBitcoinRelay(relay).finalizationParameter()`) by calling `setTransferDeadline()`.
  . Change the protocol token burning percentage fee (**0-100%**) by calling `setProtocolPercentageFee()`.
  . Change the slasher percentage fee (**0-100%**) by calling `setSlasherPercentageReward()`.
  . Change the bitcoin transaction fee (**to an arbitrarily high/low value**) by calling `setBitcoinFee()`.

The `LockersLogic.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
  . Assign a new `_owner` address by calling `transferOwnership()`.

-
    . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.

    . Add/Remove arbitrary addresses from the `minters[]` role by calling `addMinter()`/`removeMinter()`.

    . Add/Remove arbitrary addresses from the `burners[]` role by calling `addBurner()`/`removeBurner()`.

    . Setting/Unsetting the paused state for the contract (and thereby impacting calls to `selfRemoveLocker()`, `slashIdleLocker()`, `slashTheifLocker()`, `liquidateLocker()`, `buySlashedCollateralOfLocker()`, `mint()` and `burn()`) by calling `pauseLocker()`/`unPauseLocker()`.

    . Change the locker percentage fee (**0-100%**) by calling `setLockerPercentageFee()`.

    . Arbitrarily change the minimum required TeleDAO token amount for becoming a locker by calling `setMinRequiredTDTLockedAmount()`.

    . Arbitrarily change the minimum required native token amount for becoming a locker by calling `setMinRequiredTNTLockedAmount()`.

    . Change the price oracle contract address by calling `setPriceOracle()`.

    . Change the CCBurnRouter contract address by calling `setCCBurnRouter()`.

    . Change the ExchangeConnector contract address by calling `setExchangeConnector()`.

    . Change the TeleBTC token contract address by calling `setTeleBTC()`.

    . Change the lockers collateral ratio by calling `setCollateralRatio()`.

    . Arbitrarily change the lockers liquidation ration by calling `setLiquidationRatio()`.

    . Add requested lockers/Remove existing lockers by calling `addLocker()`/`ownerRemoveLocker`.

- `minters[]`, as set through `addMinter()` by `_owner`:
    . Mint an arbitrary amount (up to the capacity of the provided locker account) of TeleBTC to an arbitrary address by calling `mint()`.

- `burners[]`, as set through `addBurner()` by `_owner`:
    . Burn a provided amount of TeleBTC (up to a lockers `netMinted` amount, minus fees) by calling `burn()`.

- `ccBurnRouter`, as set through `initialize()` or `setCCBurnRouter()` by `_owner`:
    . **Transfer a lockers native tokens to an arbitrary address** by calling `slashIdleLocker()` or `slashTheifLocker()`.

The `CollateralPool.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
    . Assign a new `_owner` address by calling `transferOwnership()`.

    . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.

    . Arbitrarily change the collateralization ration (`collateralizationRatio`) in the pool by calling `setCollateralizationRatio()`.

The `CollateralPoolFactory.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
    . Assign a new `_owner` address by calling `transferOwnership()`.

    . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.

    . Create collateral pools (and add to `allCollateralPools`) by calling `createCollateralPool()`.

    . Remove collateral pools from `allCollateralPools` by calling `removeCollateralPool()`.

The `InstantPool.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
    . Assign a new `_owner` address by calling `transferOwnership()`.

    . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.

    . Assign an arbitrary address as the new instant router by calling `setInstantRouter()`.

    . Arbitrarily change the instant router fee percentage (**0%-type(uint256).max%**) by calling `setInstantPercentageFee()`.

    . Arbitrarily change the TeleBTC address by calling `setTeleBTC()`.

- `instantRouter`, as set through `constructor()` or `setInstantRouter()` by `_owner`:
    . **Transfer an arbitrary amount of TeleBTC from the contract to an arbitrary address by calling** `getLoan()`.

The `PriceOracle.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
    . Assign a new `_owner` address by calling `transferOwnership()`.

    . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.

    . Set an arbitrary address to be the connecter for any exchange router by calling `addExchangeConnector()`/`removeExchangeConnector()`.

    . Set the price proxy for any token pair to an arbitrary address by calling `setPriceProxy()`.

    . Arbitrarily change the acceptable oracle delay in seconds (**0s - type(uint256).max s**) by calling `setAcceptableDelay()`.

    . Arbitrarily change the oracle native token address by calling `setOracleNativeToken()`.

The `InstantRouter.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
    . Assign a new `_owner` address by calling `transferOwnership()`.

    . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.

    . Set/Unset the contract into/out of the paused state by calling `pause()`/`unpause()`.

    . Change the payback deadline by calling `setPaybackDeadline()`.

-
  . Change the slasher reward percentage (**0%-100%**) by calling `setSlasherPercentageReward()`.
  . Change the TeleBTC token address by calling `setTeleBTC()`.
  . Change the relay contract address by calling `setRelay()`.
  . Change the collateral pool factory contract address by calling `setCollateralPoolFactory()`.
  . Change the price oracle contract address by calling `setPriceOracle()`.
  . Change the TeleBTC instant pool contract address by calling `setTeleBTCInstantPool()`.
  . Change the default exchange connector contract address by calling `setDefaultExchangeConnector()`.

The `CCExchangeRouter.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
  . Assign a new `_owner` address by calling `transferOwnership()`.
  . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.
  . Change the relay contract address by calling `setRelay()`.
  . Change the instant router contract address by calling `setInstantRouter()`.
  . Change the lockers contract address by calling `setLockers()`.
  . Change the exchange connector contract address for any app id by calling `setExchangeConnector()`.
  . Change the TeleBTC token address by calling `setTeleBTC()`.
  . Change the protocol fee percentage (**0%-100%**) by calling `setProtocolPercentageFee()`.
  . Change the treasury address by calling `setTreasury()`.

The `CCTransferRouter.sol` contract contains the following privileged roles:

- `_owner`, as initialized during the `constructor()` execution to `msg.sender`:
  . Assign a new `_owner` address by calling `transferOwnership()`.
  . Rennounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.
  . Change the protocol fee percentage (**0%-100%**) by calling `setProtocolPercentageFee()`.
  . Change the relay contract address by calling `setRelay()`.
  . Change the lockers contract address by calling `setLockers()`.
  . Change the instant router contract address by calling `setInstantRouter()`.
  . Change the TeleBTC token address by calling `setTeleBTC()`.
  . Change the treasury address by calling `setTreasury()`.

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

Update: The client has acknowledged the issue and commented on the following: "We included additional explanations in our docs about why such roles are necessary. The owner role will eventually be handed to the community and DAO will take the control of it.", some roles and their privileges at https://docs.teleportdao.xyz/contracts/system-roles.

QSP-24 Potentially Faulty Use of `Uniswap` Interfaces

Severity: *Low Risk*

Status: Fixed

Description: The function `getInputAmount()` checks if a liquidity pool exists by checking that the following value is non-zero:

```
address liquidityPool = IUniswapV2Factory(liquidityPoolFactory).getPair(_inputToken, _outputToken);
```

However, `UniswapV2Factory` will return the correct pool regardless of the order of `_inputToken` and `_outputToken` (as in here). This affects the following check:

```
(/*reserveIn*/, uint reserveOut, /*timestamp*/) = IUniswapV2Pair(liquidityPool).getReserves();
if (_outputAmount >= reserveOut) {
    return (false, 0);
}
```

Since `_outputToken` may be associated with token0 in the `UniswapV2Pair`, this check may compare against the incorrect reserve.

Recommendation: Check that `_inputToken` corresponds to `token0` in the pair. If not, compare against `reserveIn`.

Update: Fixed as recommended.

QSP-25 Highly Utilized Pools May Be Difficult to Exit

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `InstantPool.sol`

Description: The function `removeLiquidity()` allows a provider to remove their funds from the pool. However, this function will fail if there are insufficient funds in the contract, i.e., all funds are currently on loan. If the protocol is highly utlized, the provider may have to compete with new loan requests in order to recover their funds.

Recommendation: One approach is to use a two-step process for liquidity providers removing liquidity. First, the provider can request to remove tokens, preventing new loans from utilizing those funds. When the funds become available (i.e., a previous loan is repaid), the funds would be available for the liqudity provider, as new loans would be restricted from those funds.
Note that other protocols such as Aave use the concept of a "utilization rate", making loans more costly when resources are scarce (i.e., highly utilized).

Update: The team acknowledged the issue and will plan to add utilization rate mechanism in the future.

QSP-26 Cannot Slash if Exchange Is Illiquid

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `InstantRouter.sol`, `UniswapV2Connector.sol`

Description: The function `slashUser()` invokes `IExchangeConnector(defaultExchangeConnector).getInputAmount()` to check how much `collateralToken` is needed to cover the loaned `teleBTC`. We then have the following check:

```
require(result == true, "InstantRouter: liquidity pool doesn't exist");
```

However, `UniswapV2Connector.getInputAmount()` will also return `false` when there are insufficient reserves (`teleBTC`) to cover the swap.

Recommendation: Consider cases where the pool is illiquid to complete the full swap.

Update: The team said this is intended and fixed the revert message to reflect the situation.

QSP-27 Configuration Requirements Not Strictly Enforced

Severity: *Low Risk*

Status: Fixed

File(s) affected: `LockersLogic.sol`, `InstantRouter.sol`, `BitcoinRelay.sol`

Description: The specification for lockers indicates that "the collateral ratio must be greater than the liquidation ratio". There are two related issues:

1. Although `LockersLogic._setCollateralRatio()` checks that `_collateralRatio >= liquidationRatio` (thus ensuring that the configuration is correct after invoking the `constructor`), the owner can additionally invoke `setLiquidationRatio()`. Since this function does not have a corresponding check, the contract could enter an erroneous state where `_collateralRatio < liquidationRatio`.
2. The conditional is not strictly enforced as `LockersLogic._setCollateralRatio()` checks that `_collateralRatio >= liquidationRatio` (i.e., using `>=` instead of `>`, as suggested by the specification).
3. The `LockersLogic. constructor()` checks that `_minRequiredTDTLockedAmount != 0 || _minRequiredTNTLockedAmount != 0`, however the owner could call the setter functions `setMinRequiredTDTLockedAmount()` and `setMinRequiredTNTLockedAmount()` to set both values to zero.
4. In `InstantRouter._setPaybackDeadline()`, we ensure that `_paybackDeadline > _finalizationParameter`. However, if we later use `BitcoinRelay.setFinalizationParameter()` to increase this parameter, the invariant could be violated.

Recommendation: Add additional checks to the private setter functions. Either change the conditional or re-word the specification.

Update: The fixes are applied accordingly. Additionally, the team mentioned that `_minRequiredTDTLockedAmount` can be zero hence related checks are removed for it.

QSP-28 Relayer May Not Be Compensated

Severity: *Low Risk*

Status: Acknowledged

Description: In the function `_sendReward()` of `BTCRelayer.sol`, it can be clearly seen that when there are not enough rewards (either the native token or the dao token), the transfer of the funds are being skipped and not accounted anywhere. Since the fee charging mechanism is using estimation from last epoch and not necessarily precise, it is possible that the contract may not have enough assets to provide payout and by its logic skipping the payouts to the relayer.

Recommendation: Clarify how should the protocol handle this scenario and make it clear to the relayers about this risk so that they could monitor. Alternatively, consider another design that does not skip over the reward payout.

Update: The team replied that it is acceptable when the system does not reward the relayers as they will be committed to running one in that case. Note that this means the external relayers need to take note whether rewards are actually been paid out.

QSP-29 Estimations of Fee May Change Unintentionally by Changing `epochLength`

Severity: *Low Risk*

Status: Acknowledged

Description: `epochLength` is an admin controlled storage. It is used to estimate how much fee should the contract charge users when they are interacting with it through the `checkTxProof()`. Suppose that the admin have decided to 2x the length of an epoch. The `lastEpochQueries` would remain the same, but the fee would have gone 2x due to the formula in `_calculateFee()`.

Recommendation: Clarify whether this is acceptable. If not, consider adding a variable `lastEpochLength` and use it in the fee calculation instead to tackle the corner case.

Update: The team acknowledged the issue and view the temporary incorrect fee estimation as an acceptable outcome.

QSP-30 Unnecessary Precision Loss Due to Suboptimal Arithmetic Order

Severity: *Low Risk*

Status: Fixed

File(s) affected: `PriceOracle`

Description: The price calculation in `PriceOracle. _equivalentOutputAmountFromOracle()` faced some loss in precision as division was performed before multiplication for calculating a particular value.

Update: The team fixed it by changing the order of calculation.

QSP-31 Block Timestamp Manipulation

Severity: *Informational*

Status: Acknowledged

File(s) affected: `locker/LockerLogic.sol`, `routers/InstantRouter.sol`

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that validators individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their purposes. If a smart contract relies on a timestamp, it must take this into account.
In the function `instantCCTransfer`, the deadline is checked against the timestamp. This relies on `block.timestamp` to be correct, but that aspect can be manipulated by validators/attackers for their purposes by up to 900 seconds.

Recommendation: Clarify the impact in the given protocol design in user-facing documentation and, if necessary, use an oracle for time inquiries.

Update: The client has acknowledged the issue and commented on the following: "900 seconds does not have an impact for our purposes in the mentioned function. Since even after small delays the malicious user gets slashed eventually."

QSP-32 Unlocked Pragma

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/*`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term “unlocked”.

Recommendation: For consistency and to prevent unexpected behaviour in the future, we recommend removing the caret to lock the file onto a specific Solidity version.

Update: the team has fixed the issue by locking the solidity version within the range of `0.8.0` and `0.8.4`.

QSP-33 Application Monitoring Can Be Improved by Emitting More Events

Severity: *Informational*

Status: Mitigated

File(s) affected: `erc20/TeleBTC.sol`, `relay/BitcoinRelay.sol`, `routers/CCBurnRouter.sol`, `lockers/LockersLogic.sol`, `pools/CollateralPool.sol`, `pools/InstantPool.sol`, `oracle/PriceOracle.sol`, `routers/InstantRouter.sol`, `routers/CCTransferRouter.sol`

Description: To validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Furthermore, any important state transition can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs, or hacks. Below, we present a non-exhaustive list of events that could be emitted to improve the application management:

1. `TeleBTC.addMinter()` does not emit an event reflecting changes made to the state variable `minters[]`.
2. `TeleBTC.removeMinter()` does not emit an event reflecting changes made to the state variable `minters[]`.
3. `TeleBTC.addBurner()` does not emit an event reflecting changes made to the state variable `burners[]`.
4. `TeleBTC.removeBurner()` does not emit an event reflecting changes made to the state variable `burners[]`.
5. `BitcoinRelay.setRewardAmountInTDT()` does not emit an event reflecting changes made to the state variable `rewardAmountInTDT`.
6. `BitcoinRelay.setFinalizationParameter()` does not emit an event reflecting changes made to the state variable `finalizationParameter`.
7. `BitcoinRelay.setRelayerPercentageFee()` does not emit an event reflecting changes made to the state variable `relayerPercentageFee`.
8. `BitcoinRelay.setEpochLength()` does not emit an event reflecting changes made to the state variable `epochLength`.
9. `BitcoinRelay.setBaseQueries()` does not emit an event reflecting changes made to the state variable `baseQueries`.
10. `BitcoinRelay.setSubmissionGasUsed()` does not emit an event reflecting changes made to the state variable `submissionGasUsed`.
11. `BitcoinRelay.checkTxProof()` does not emit an event reflecting changes made to the state variable `currentEpochQueries`.
12. `BitcoinRelay._addToChain()` does not emit an event reflecting changes made to the state variable `chain[]`.
13. `CCBurnRouter.setRelay()` does not emit an event reflecting changes made to the state variable `relay`.
14. `CCBurnRouter.setLockers()` does not emit an event reflecting changes made to the state variable `lockers`.
15. `CCBurnRouter.setTeleBTC()` does not emit an event reflecting changes made to the state variable `teleBTC`.
16. `CCBurnRouter.setTransferDeadline()` does not emit an event reflecting changes made to the state variable `transferDeadline`.
17. `CCBurnRouter.setProtocolPercentageFee()` does not emit an event reflecting changes made to the state variable `protocolPercentageFee`.
18. `CCBurnRouter.setSlasherPercentageReward()` does not emit an event reflecting changes made to the state variable `slasherPercentageReward`.
19. `CCBurnRouter.setBitcoinFee()` does not emit an event reflecting changes made to the state variable `bitcoinFee`.
20. `CCBurnRouter._updateIsUsedAsBurnProof()` does not emit an event reflecting changes made to the state variable `isUsedAsBurnProof[]`.
21. `LockersLogic.addMinter()` does not emit an event reflecting changes made to the state variable `minters[]`.
22. `LockersLogic.removeMinter()` does not emit an event reflecting changes made to the state variable `minters[]`.
23. `LockersLogic.addBurner()` does not emit an event reflecting changes made to the state variable `burners[]`.
24. `LockersLogic.removeBurner()` does not emit an event reflecting changes made to the state variable `burners[]`.
25. `LockersLogic.setLockerPercentageFee()` does not emit an event reflecting changes made to the state variables `lockerPercentageFee` and `libParams.lockerPercentageFee`.
26. `LockersLogic.setMinRequiredTDTLockedAmount()` does not emit an event reflecting changes made to the state variables `minRequiredTDTLockedAmount` and `libParams.minRequiredTDTLockedAmount`.
27. `LockersLogic.setMinRequiredTNTLockedAmount()` does not emit an event reflecting changes made to the state variables `minRequiredTNTLockedAmount` and `libParams.minRequiredTNTLockedAmount`.

28. `LockersLogic.setPriceOracle()` does not emit an event reflecting changes made to the state variables `priceOracle` and `libParams.priceOracle`.
29. `LockersLogic.setCCBurnRouter()` does not emit an event reflecting changes made to the state variables `ccBurnRouter` and `libParams.ccBurnRouter`.
30. `LockersLogic.setExchangeConnector()` does not emit an event reflecting changes made to the state variables `exchangeConnector` and `libParams.exchangeConnector`.
31. `LockersLogic.setTeleBTC()` does not emit an event reflecting changes made to the state variables `teleBTC` and `libParams.teleBTC`.
32. `LockersLogic.setCollateralRatio()` does not emit an event reflecting changes made to the state variables `collateralRatio` and `libParams.collateralRatio`.
33. `LockersLogic.setLiquidationRatio()` does not emit an event reflecting changes made to the state variables `liquidationRatio` and `libParams.liquidationRatio`.
34. `CollateralPool.setCollateralizationRatio()` does not emit an event reflecting changes made to the state variable `collateralizationRatio`.
35. `InstantPool.setInstantRouter()` does not emit an event reflecting changes made to the state variable `instantRouter`.
36. `InstantPool.setInstantPercentageFee()` does not emit an event reflecting changes made to the state variable `instantPercentageFee`.
37. `InstantPool.setTeleBTC()` does not emit an event reflecting changes made to the state variable `teleBTC`.
38. `PriceOracle.setAcceptableDelay()` does not emit an event reflecting changes made to the state variable `acceptableDelay`.
39. `PriceOracle.setOracleNativeToken()` does not emit an event reflecting changes made to the state variable `oracleNativeToken`.
40. `InstantRouter.setPaybackDeadline()` does not emit an event reflecting changes made to the state variable `paybackDeadline`.
41. `InstantRouter.setSlasherPercentageReward()` does not emit an event reflecting changes made to the state variable `slasherPercentageReward`.
42. `InstantRouter.setTeleBTC()` does not emit an event reflecting changes made to the state variable `teleBTC`.
43. `InstantRouter.setRelay()` does not emit an event reflecting changes made to the state variable `relay`.
44. `InstantRouter.setCollateralPoolFactory()` does not emit an event reflecting changes made to the state variable `collateralPoolFactory`.
45. `InstantRouter.setPriceOracle()` does not emit an event reflecting changes made to the state variable `priceOracle`.
46. `InstantRouter.setTeleBTCInstantPool()` does not emit an event reflecting changes made to the state variable `teleBTCInstantPool`.
47. `InstantRouter.setDefaultExchangeConnector()` does not emit an event reflecting changes made to the state variable `defaultExchangeConnector`.
48. `CCExchangeRouter.setRelay()` does not emit an event reflecting changes made to the state variable `relay`.
49. `CCExchangeRouter.setInstantRouter()` does not emit an event reflecting changes made to the state variable `instantRouter`.
50. `CCExchangeRouter.setLockers()` does not emit an event reflecting changes made to the state variable `lockers`.
51. `CCExchangeRouter.setTeleBTC()` does not emit an event reflecting changes made to the state variable `teleBTC`.
52. `CCExchangeRouter.setProtocolPercentageFee()` does not emit an event reflecting changes made to the state variable `protocolPercentageFee`.
53. `CCExchangeRouter.setTreasury()` does not emit an event reflecting changes made to the state variable `treasury`.
54. `CCTransferRouter.setProtocolPercentageFee()` does not emit an event reflecting changes made to the state variable `protocolPercentageFee`.
55. `CCTransferRouter.setRelay()` does not emit an event reflecting changes made to the state variable `relay`.
56. `CCTransferRouter.setLockers()` does not emit an event reflecting changes made to the state variable `lockers`.
57. `CCTransferRouter.setInstantRouter()` does not emit an event reflecting changes made to the state variable `instantRouter`.
58. `CCTransferRouter.setTeleBTC()` does not emit an event reflecting changes made to the state variable `teleBTC`.
59. `CCTransferRouter.setTreasury()` does not emit an event reflecting changes made to the state variable `treasury`.

Recommendation: Consider emitting the events.

Update: 1. Fixed.

1. Fixed.
2. Fixed.
3. Fixed.
4. Fixed.
5. Fixed.
6. Fixed.
7. Fixed.
8. Fixed.
9. Fixed.
10. **Unresolved**.
11. **Unresolved**/False positive (debatable, as the existing event does not emit all changes)
12. Fixed.
13. Fixed.
14. Fixed.
15. Fixed.
16. Fixed.
17. Fixed.
18. Fixed.
19. **Unresolved**.
20. Fixed.
21. Fixed.

22. Fixed.
23. Fixed.
24. Fixed.
25. Fixed.
26. Fixed.
27. Fixed.
28. Fixed. (They now emit there 3 events...)
29. Fixed.
30. Fixed.
31. Fixed.
32. Fixed.
33. Fixed.
34. Fixed.
35. Fixed.
36. Fixed.
37. Fixed.
38. Fixed.
39. Fixed.
40. Fixed.
41. Fixed.
42. Fixed.
43. Fixed.
44. Fixed.
45. Fixed.
46. Fixed.
47. Fixed.
48. Fixed.
49. Fixed.
50. Fixed.
51. Fixed.
52. Fixed.
53. Fixed.
54. Fixed.
55. Fixed.
56. Fixed.
57. Fixed.
58. Fixed.

QSP-34 Checks-Effects-Interactions Pattern Violations

Severity: *Informational*

Status: Mitigated

File(s) affected: `contracts/relay/BitcoinRelay.sol`, `contracts/lockers/LockersLogic.sol`, `contracts/pools/CollateralPool.sol`, `contracts/pools/InstantPool.sol`

Description: As a best practice and to prevent unwanted external contract side effects, it is advised to adhere to the "Checks-Effects-Interactions"-pattern, even in the presence of reentrancy guards.
The following instances were observed where said pattern was violated:

1. `BitcoinRelay.checkTxProof()`: Sends native tokens back to `msg.sender`, before modifying `currentEpochQueries` (Function is protected by `nonReentrant`).
2. `BitcoinRelay._pruneChain()`: Sends native and DAO tokens back to `msg.sender` in a sub-call to `_sendReward()`, before emitting `BlockFinalized` (Externally public function is protected by `nonReentrant`).
3. `BitcoinRelay._addToChain()`: Sends native and DAO tokens back to `msg.sender` in a sub-call to `_sendReward()`, before modifying `lastEpochQueries` and `currentEpochQueries` in a sub-call to `_updateFee` (Externally public function is protected by `nonReentrant`).
4. `BitcoinRelay._addHeaders()`: Sends native and DAO tokens back to `msg.sender` in a sub-call to `_sendReward()`, before modifying `_previousHash` and emitting `BlockAdded` (Externally public function is protected by `nonReentrant`).
5. `LockersLogic.requestToBecomeLocker()`: Transfers native and DAO tokens tokens to the contract `msg.sender`, before modifying `lockersMapping[]` and `totalNumberOfCandidates` (Function is protected by `nonReentrant`).
6. `LockersLogic.liquidateLocker()`: Performs calls to external contracts (`ICCBurnRouter(ccBurnRouter).ccBurn()` and its sub-calls), before modifying `lockersMapping[]` (Function is protected by `nonReentrant`).
7. `CollateralPool.addCollateral()`: Performs calls to external contracts (`IERC20(collateralToken).transferFrom()`), before modifying state variables through `_mint()` (Function is protected by `nonReentrant`).

8. `InstantPool.addLiquidity()`: Performs calls to external contracts (`IERC20(teleBTC).transferFrom()`), before modifying state variables through `_mint()` and `totalAddedTeleBTC` (Function is protected by `nonReentrant`).
9. `InstantPool.addLiquidityWithoutMint()`: Performs calls to external contracts (`IERC20(teleBTC).transferFrom()`), before modifying state variable `totalAddedTeleBTC` (Function is protected by `nonReentrant`).
10. `InstantPool.removeLiquidity()`: Performs calls to external contracts (`IERC20(teleBTC).transfer()`), before modifying state variables through `_burn()` (Function is protected by `nonReentrant`).

Recommendation: Consider re-structuring the code, such that it no longer violates the "Checks-Effects-Interactions"-pattern and/or add reentrancy guards at corresponding calling locations, if not already present.

Update: 1. Fixed.

1. **Unresolved**.
2. Fixed.
3. False positive - To be removed.
4. **Unresolved** (`totalNumberOfCandidates` still modified after transfers).
5. Fixed.
6. **Unresolved**.
7. **Unresolved**.
8. **Unresolved**.
9. Fixed.

QSP-35 Timestamp Potentially Overflowable in `PriceOracle.equivalentOutputAmount()`

Severity: *Informational*

Status: Fixed

File(s) affected: `contracts/oracle/PriceOracle.sol`

Description: In function `PriceOracle.equivalentOutputAmount()` an unsafe cast operation is used to cast variable `timestamp` (of type `uint256`) to type `int256`, exposing it to overflows, when reaching values higher than `type(int256).max`.

Recommendation: We recommend the use of a safe cast library, or manually enforcing `timestamp` to be within the bound of `type(int256).max`.

Update: the team has fixed the issue by replacing the native cast operation with its safe alternative (`.toInt256()`), as suggested.

QSP-36 Clone-and-Own

Severity: *Informational*

Status: Fixed

File(s) affected: `libraries/SafeMath.sol`, `erc20/ERC20.sol`, `erc20/WETH.sol`, `erc20/Context.sol`

Description: The clone-and-own approach involves copying and adjusting open-source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve traceability of the file.

Update: the team has fixed the issue by removing said cloned contracts and instead using them through a package manager, as suggested.

QSP-37 Remove-Loops Gas-Optimizable

Severity: *Informational*

Status: Fixed

File(s) affected: `pools/CollateralPoolFactory.sol`, `oracle/PriceOracle.sol`, `routers/InstantRouter.sol`

Description: Functions `CollateralPoolFactory._removeElement()`, `PriceOracle._removeElementFromExchangeRoutersList` and `InstantRouter._removeElement()` unnecessarily shift all `[index, length]` elements by one, when removing one element from the `allCollateralPools[]`/`exchangeRoutersList[]`/`instantRequests[]` arrays. Depending on the number of elements in the array and the position of the to-be-removed element, this operation will unnecessarily burn gas.

Recommendation: Consider just swapping the to-be-removed element with the last element in the array and then remove it (`.pop()`).

Update: the team has fixed the issue by implementing the suggested replace-and-pop method. The algorithm for `InstantRouter._removeElement()` remains the same as it was acknowledged by the developers to be dependent on the order.

QSP-38 Commented-Out Code

Severity: *Informational*

Status: Mitigated

File(s) affected: `erc20/ERC20.sol`, `erc20/WETH.sol`, `relay/BitcoinRelay.sol`, `pools/InstantPool.sol`

Description: The smart contract contains commented-out code. It is unclear whether this is intentional or not. Furthermore, it makes maintenance and verification of the code cumbersome. The following instances have been identified:

1. `ERC20.sol#L6`: `//import "./Context.sol";`.
2. `ERC20.sol#L243`: `// require(account != address(0), "ERC20: mint to the zero address");`.
3. `WETH.sol#L10`: `// using SafeMath for uint;`.
4. `BitcoinRelay.sol#L537`: `// bytes29 header = chain[currentHeight][stableIdx];`.
5. `InstantPool.sol#L130`: `// totalAddedTeleBTC = totalAddedTeleBTC + instantFee;`.

Recommendation: We recommend removing the commented-out code from the contract.

Update: 1. Obsolete, due to fix in QSP-17 "Clone-and-Own".

1. Obsolete, due to fix in QSP-17 "Clone-and-Own".
2. Fixed.
3. **Unresolved**.
4. Fixed.

QSP-39 Unnecessary Use of `SafeMath` in Solidity 0.8.x

Severity: *Informational*

Status: Fixed

File(s) affected: `erc20/WETH.sol`, `erc20/ERC20.sol`

Description: Solidity 0.8.x has a built-in mechanism for dealing with overflows and underflows. There is no need to use the `SafeMath` library (it only increases gas usage).

Recommendation: We recommend against using `SafeMath` in Solidity 0.8.x.

Update: the team has fixed the issue by removing the import and use of the `SafeMath` library, as suggested.

QSP-40 No Incentive for Teleporter to Slash User when Locked Collateral Is Lower than the Loan Amount

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `routers/InstantRouter.sol`

Description: When a user doesn't pay back the loan before the deadline, another user can slash the defaulter's collateral. This works when collateral is above the loan amount, as another user has an incentive to slash and get a reward.
While, if the collateral is below the loan amount, there is no incentive for the other user to slash. Hence, there would be cases where this collateral is just locked there.

Recommendation: Consult if this is the expected behaviour and add extra incentives for the cases where collateral falls below the loan amount.

Update: The client has acknowledged the issue and commented on the following: "Anyone can call the slashing function in the contract. In the mentioned case, the instant pool providers have the incentive to slash the user."

QSP-41 `TypedMemView.encodeHex()` Always Reverts

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/libraries/TypedMemView.sol`

Description: Contract `TypedMemView.sol` is a fork with custom changes. One of the main changes was dropping `SafeMath.sol`, in favour of upgrading to a solidity pragma version `^0.8.0` (as it implicitly adds SafeMath's arithmetic overflow checks). In `TypedMemView.sol#L137` (function `encodeHex()`) the code purposefully abused the previously unchecked arithmetic underflow, when iterating over a loop (also as indicated by a comment):

```
        // abusing underflow here =_=
        for (uint8 i = 15; i < 255 ; i -= 1) {
```

Due to said use of a solidity version equal or higher than `0.8.0` and its implicit overflow and underflow checks, this code will always revert, when trying to underflow/decrease loop variable `i` from zero to `255`, preventing any successful execution of this function.

Note: This seemed to remain unnoticed as this function is not covered in any tests. We therefore further strongly recommend extending the test suite.

Recommendation: Consider wrapping said for-loop within an `unchecked {...}` block.

Update: the team has fixed the issue by wrapping said loop inside an `unchecked {...}` block, as suggested.

QSP-42 Length Overflowable in `BitcoinHelper.revertNonMinimal()`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `contracts/libraries/BitcoinHelper.sol`

Description: Function `BitcoinHelper.revertNonMinimal()` casts the length of the provided memory view parameter `ref` down from `uint96` to `uint8` using the primitive and unsafe cast operation `uint8(ref.len())`, which is prone to overflows. Passing a memory view `ref` with a length greater than 255 will therefore lead to overflows, wrapping around to a value between 0-255, which in turn will lead to a wrong value being parsed and displayed in the error message.

Recommendation: We recommend replacing the unsafe cast operation `uint8(...)` with i.e. OpenZeppelins SafeCast library equivalent `toUint8(...)`, or perform a manual size check (i.e. `require(ref.len() <= 255, ...);`.

Update: the team has fixed by replacing the native cast operation with its safe alternative (`.toUint8()`), as suggested.

Automated Analyses

Slither

The majority of the issues alerted by slither are false positive.

Code Documentation

1. In `LockerLogic.sol`, the description for `getLockerCapacity()` is incorrect: "Get how much net this locker has minted".
2. Are the constants `200 * 10 ** 8` and `2000` in the constructor documented anywhere?
3. In `InstantRouter.sol`, "Transfes" should be "Transfers".

Adherence to Best Practices

1. A few TODOs are left in the code LockerLogic.sol:L468 There is a typo in the function name of the `slashTheifLocker()` in `LockerLogic.sol:L570` (-> `slashThiefLocker()`)
2. The following typographical errors have been noted:
    1. `.solcover.js#28`: `ture` -> `true`.
    2. `TypedMemView.sol#L385`: `encoded loc` -> `encoded len`.
    3. `TypedMemView.sol#L462`: `byte` -> `bytes`.
    4. `TypedMemView.sol#L537`: `have` -> `has`.
    5. `BitcoinHelper.sol#L85`: `integer as` -> `integer length as`.
    6. `BitcoinRelay.sol#L33`: `parnet` -> `parent`.
    7. `BitcoinRelay.sol#L93` and `L101`: `an` -> `a`.
    8. `BitcoinRelay.sol#L577`: `header` -> `headers`.
    9. `BitcoinRelay.sol#L305-306`: `addHeaders` -> `addHeaders and ownerAddHeaders`.
    10. `BitcoinRelay.sol#L313-315`: `addHeadersWithRetarget` -> `addHeadersWithRetarget and ownerAddHeadersWithRetarget`.
    11. `CCBurnRouter.sol#L108`: `shoudl` -> `should`.
    12. `CCBurnRouter.sol#L173`, `L178-L180`, `L528`, `L609`, `L619` and `L627`: `Remained`/`remained` -> `Remaining`/`remaining`.
    13. `CCBurnRouter.sol#L277`: `paid` -> `pay`.
    14. `CCBurnRouter.sol#L279`: `that their` -> `whose`.
    15. `CCBurnRouter.sol#L280` and `L340`: `successfull` -> `successful`.
    16. `LockersLogic.sol#L983`, `L990`, `L991`, `L995-997`: `_removingLokcer` -> `_removingLocker`.
    17. `LockersLogic.sol#L621`: `which its` -> `whose`.
    18. `LockersLogic.sol#L624` and `L708`: `is intend` -> `intends`.
    19. `LockersLogic.sol#L709`: `is` -> `if`.
    20. `LockersLogic.sol#L846`: Spurious `got`.
    21. `LockersLogic.sol#L894`: Spurious `the`.
    22. `CollateralPoolFactory.sol#L34`: Spurious `that`.
    23. `PriceProxy.sol#L19`: `exchnage` -> `exchange`.
    24. `ICCBurnRouter.sol#L32`: `Toral` -> `Total`.
    25. `InstantRouter.sol#L43`: `contrac` -> `contract`.
    26. `InstantRouter.sol#L45`: `Dealine` -> `Deadline`.
3. The code comment in `TypedMemView.sol#L361` states `shift out lower 24 bytes`. However, more precisely it additionally shifts out the implicit trailing 3 zero bytes (Further explaining the shift count of `216 = (12 + 12 + 3) * 8`).
4. Missing or incorrect NatSpec comments:
    1. `TeleBTC.isMinter()`: Missing NatSpec comment for parameter `account`.
    2. `TeleBTC.isBurner()`: Missing NatSpec comment for parameter `account`.
    3. `BitcoinRelay.findAncestor()`: Missing NatSpec comment for parameter `_offset`.
    4. `BitcoinRelay.checkTxProof()`: Imprecise/Ambiguous NatSpec comments for parameters `_intermediateNodes` and `_index`).
    5. `BitcoinRelay._getFee()`: Missing NatSpec comment for parameter `gasPrice`.
    6. `BitcoinRelay._sendReward()`: Missing NatSpec comment for parameter `_height`.
    7. `BitcoinRelay._sendReward()`: Incorrect NatSpec comments for return values (Return values are of type `uint256` and not one boolean value).
    8. `CCBurnRouter.ccBurn()`: Missing NatSpec comment for parameter `_scriptType`.
    9. `CCBurnRouter._getFees()`: Incorrect NatSpec comments for parameters `_amount` and `_lockerTargetAddress`.
    10. `CCBurnRouter._saveBurnRequest()`: Missing NatSpec comment for parameter `_scriptType`.
    11. `DataTypes.locker`: Missing NatSpec comment for structure element `isScriptHash`.

12. `LockersLogic.sol#L175-L176`, `L182-L183`, `L193`, `L201`, `L208` and `L948`: Empty NatSpec comment entries.
13. `LockersLogic.isLocker()`: Wrong NatSpec parameter name and description for parameter `_lockerLockingScript` and wrong `@notice` description (Parameter is a script, not the direct address itself).
14. `LockersLogic.setMinRequiredTDTLockedAmount()` and `LockersLogic.setMinRequiredTNTLockedAmount()` share the same NatSpec comments. Consider adding their corresponding token dependence (TeleportDAO token and native token), to better differentiate.
15. `LockersLogic.setExchangeConnector()`: Wrongfully states `and updates wrapped avax addresses`.
16. `LockersLogic.slashIdleLocker()`: Incorrect NatSpec parameter name for parameter `_rewardRecipient`.
17. `LockersLogic.priceOfOneUnitOfCollateralInBTC()`: Missing NatSpec comment for return parameter.
18. `LockersLogic._isMinter()`: Missing NatSpec comment for return parameter.
19. `LockersLogic._isBurner()`: Missing NatSpec comment for return parameter.
20. `IInstantPool.InstantLoan()`: Incorrect NatSpec comment at keyword `@notice` (no collateral is added, rather TeleBTC tokens are transfered away).
21. `PriceOracle.constructor()`: Missing NatSpec comment for parameter `_oracleNativeToken`.
22. `ICCBurnRouter.burnRequest`: Missing NatSpec comment for struct element `scriptType`.
23. `ICCBurnRouter.CCBurn()`: Missing NatSpec comment for event parameter `scriptType`.
24. `IInstantRouter.instantRequest`: Swapped order of NatSpec struct element comments `collateralToken` and `paybackAmount`.
25. `InstantRouter.setRelay()`, `InstantRouter.setCollateralPoolFactory()`, `InstantRouter.setPriceOracle()` and `InstantRouter.setTeleBTCInstantPool()`: Incorrect NatSpec comment for keyword `@notice` and parameter (copy-and-paste from `InstantRouter.setTeleBTC()`).
26. `RequestHelper.*()`: Missing NatSpec comment for return parameter.
27. `CCTransferRouter.setLockers()`: Incorrect NatSpec comment for keyword `@notice` (copy-and-paste from `CCTransferRouter.setRelay()`).

5. Internal state variables `BitcoinRelay.previousBlock` and `BitcoinRelay.blockHeight` are wrongfully grouped in the `// Public variables` block.

Adherence to Best Practices

1. Before rolling out code in production, any pending `TODO` items in code should be resolved in order to not deploy potentially unfinished code. In this regard the following `TODO` items still remain in code and should be resolved:
    1. `UniswapV2Connector.sol#L232`: `TODO: un-comment on production`.
    2. `TeleBTC.sol#L87`: `TODO: remove it`.
    3. `LockersLogic.sol#L468`: `TODO: adding more fields to this event`.
    4. `SafeMath.sol#L20`: `TODO: edit it`.
    5. `ERC20.sol#L242`: `FIXME: un-comment next line`.
    6. `TypedMemView.sol#L177`: `ugly. redo without assembly?`.
    7. `TypedMemView.sol#L522`: `FIXME: why the following lines need 'unchecked'`. (Note: Without `unchecked` the multiplication `uint8 bitLength = _bytes * 8;` would overflow the maximal value of type `uint8` of 255 when `_bytes` is 32 to 256)
    8. `IBitcoinRelay.sol#L87`: `see if it's needed`. (Note: Does not seem to be used)
    9. `IBitcoinRelay.sol#L89`: `see if it's needed`. (Note: Does not seem to be used)
    10. `BitcoinRelay.sol#L623-624`: `is this correct? it was in the original code, and we are not sure why is it this way`.
    11. `PriceOracle.sol#L77`: `note: we assume that the decimal of exchange returned result is _outputDecimals. Is that right?`.

2. To improve readability and lower the risk of introducing errors when making code changes, it is advised to not use magic constants throughout code, but instead declare them once (as constant and commented) and use these constant variables instead. Following instances should therefore be changed accordingly:
    1. `BitcoinHelper.sol#L15`: `2 * 7 * 24 * 60 * 60` (Consider the use of solidity built-in keyword `weeks`).
    2. `BitcoinHelper.sol#L383`: `0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff` (Consider using the existing constant `TypedMemView.NULL`).
    3. `BitcoinHelper.sol#L148`, `L219`, `L246`, `L286`, `L321`, `L365`, `L422`, `L432` and `BitcoinRelay.sol#L51`, `L229`, `L240`, `L241`, `L259`, `L260`, `L261`, `L275`, `L276`, `L295`, `L296`, `L297`: `0` (Consider using `*.ref(BTCTypes.Unknown)` instead of `*.ref(0)` in these cases).
    4. `BitcoinHelper.sol#L719-720`: `2`.
    5. `BitcoinRelay.sol#L66`, `L74`, `L439`, `L605`: `2016` (Consider re-using `BitcoinHelper.RETARGET_PERIOD_BLOCKS` instead).
    6. `BitcoinRelay.sol#L106`, `L388` and `L467`: `100`.
    7. `InstantRouter.sol#L509`: `10000`.

3. To improve readability and maintainability, it is recommended to use meaningful names when naming variables, functions, ... In this regard, consider renaming following instances:
    1. Function name `BitcoinHelper.revertBytes32()`: Consider renaming it to i.e. `reverseBytes32()` (and accordingly its NatSpec comments), as it reverses the bytes and the keyword `revert` is already reserved for a different functionality in Solidity (Further, consider the use of the existing function `TypedMemView.reverseUint256()`, with corresponding casting).
    2. Parameter name `bytes`: `TypedMemView.sol#L512`, `L540`, `L551`

4. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
    1. `ILockers.LockerSlashed()`,
    2. `ILockers.LockerLiquidated()`,
    3. `ILockers.LockerSlashedCollateralSold()`,
    4. `ILockers.MintByLocker()`,
    5. `ICollateralPoolFactory.CreateCollateralPool()`,
    6. `ICollateralPoolFactory.RemoveCollateralPool()`,

7. `IPriceOracle.ExchangeConnectorAdded()`,
8. `IPriceOracle.ExchangeConnectorRemoved()`,
9. `IPriceOracle.SetPriceProxy()`,
10. `ICCBurnRouter.LockerDispute()`,
11. `ICCExchangeRouter.CCExchange()`,
12. `ICCExchangeRouter.FailedCCExchange()`,
13. `ICCExchangeRouter.SetExchangeConnector()`,
14. `ICCTransferRouter.CCTransfer()`,
15. `IInstantRouter.InstantTransfer()`,
16. `IInstantRouter.InstantExchange()`,
17. `IInstantRouter.SlashUser()`,

5. According to best practices explicit types/type widths should be used. In this regard, consider changing occurences of `uint` to `uint256` (several files are impacted).
6. To improve readability and maintainability, it is recommended to re-use similar code blocks. In this regard, consider the following instances:
    1. `BitcoinRelay.sol#L106`, `L388` and `467`: Share the mostly similar formula `submissionGasUsed * gasPrice * (100 + relayerPercentageFee) * epochLength) / lastEpochQueries / 100`.
    2. `LockersLogic.sol`: State variables `TeleportDAOToken`, `exchangeConnector`, `priceOracle`, `minRequiredTDTLockedAmount`, `minRequiredTNTLockedAmount`, `collateralRatio`, `liquidationRatio`, `lockerPercentageFee` and `priceWithDiscountRatio` their corresponding equivalents in `libParams` share the same information, making one of them redundant. Consider removing one of them.

7. For improved readability and maintainability it is advised to use descriptive function/variable/... names, as well as descriptive error messages. In this regard, consider the following cases:
    1. `LockersLogic.sol#L384-L387` and `L419-L422`: Consider a more descriptive error message for the `require(...)` statement (i.e. `Lockers: not a candidate`).

8. In `TeleBTC.mint()`, the check `_amount <= maxmimumMintLimit` is redundant to `checkAndReduceMintLimit(_amount) == true` (and further the `== true` is not necessary).
9. In `InstantPool.getLoan()`, the constant 10000 is used instead of `MAX_INSTANT_PERCENTAGE_FEE`.
10. The event `IInstantRouter.NewDeafultExchangeConnector` is misspelled on the word "Default".
11. In `InstantRouter._lockCollateral()`, the constant `MAX_SLASHER_PERCENTAGE_REWARD` is used to normalize `collateralizationRatio`. Although it's value `10000` is correct, the name `MAX_SLASHER_PERCENTAGE_REWARD` is irrelevant to the function, and a different constant such as `ONE_HUNDRED_PERCENT` would be more appropriate.

Test Results

Test Suite Results

npx hardhat test

```
Bitcoin Relay
  Submitting block headers
     ✓ check the owner
     ✓ submit old block headers
     ✓ revert a block header with wrong PoW
     ✓ revert a block header with wrong previous hash
     ✓ submit a block header for a new epoch with same target (addHeaders)
     ✓ submit a block header with new target (addHeaders => unsuccessful)
     ✓ submit a block header with new target
  Submitting block headers with forks
     ✓ successfully create a fork
     ✓ not be able to submit too old block headers to form a fork
     ✓ successfully prune the chain
     ✓ successfully emit FinalizedBlock
  Unfinalizing a finalized block header
     ✓ unfinalize block 478559 and finalize block 478559"
  Check tx inclusion
     ✓ errors if the smart contract is paused
     ✓ transaction id should be non-zero
     ✓ errors if the requested block header is not on the relay (it is too old)
     ✓ check transaction inclusion -> when included
     ✓ reverts when enough fee is not paid
     ✓ check transaction inclusion -> when not included
     ✓ reverts when tx's block is not finalized
  #constructor
     ✓ errors if the caller is being an idiot
     ✓ errors if the period start is in wrong byte order
     ✓ stores genesis block info
  #pauseRelay
     ✓ errors if the caller is not owner
  #unpauseRelay
     ✓ errors if the caller is not owner
  #getBlockHeaderHash
     ✓ views the hash correctly
  ## Setters
     ✓ #setRewardAmountInTDT
     ✓ setRewardAmountInTDT owner check
     ✓ #setFinalizationParameter
     ✓ setFinalizationParameter owner check
     ✓ #setRelayerPercentageFee
     ✓ setRelayerPercentageFee owner check
     ✓ #setEpochLength
     ✓ setEpochLength owner check
     ✓ #setBaseQueries
     ✓ setBaseQueries owner check
     ✓ #setSubmissionGasUsed
     ✓ setSubmissionGasUsed owner check
  #addHeaders
     ✓ errors if the smart contract is paused
     ✓ errors if the anchor is unknown
     ✓ errors if it encounters a retarget on an external call
     ✓ errors if the header array is not a multiple of 80 bytes
     ✓ errors if a header work is too low
     ✓ errors if the target changes mid-chain
     ✓ errors if a prevhash link is broken
     ✓ appends new links to the chain and fires an event
     ✓ contract has no TNT but doesn't revert when paying a relayer
     ✓ contract has no TNT but has some TDT so rewards relayer only in TDT
     ✓ fails in sending reward in TDT but submission goes through successfully
     ✓ contract has enough TNT so pays the relayer
     ✓ skips some validation steps for known blocks
```

```
  #addHeadersWithRetarget
     ✓ errors if the smart contract is paused
     ✓ errors if the old period start header is unknown
     ✓ errors if the old period end header is unknown
     ✓ errors if the provided last header does not match records
     ✓ errors if the start and end headers are not exactly 2015 blocks apart
     ✓ errors if the retarget is performed incorrectly
     ✓ appends new links to the chain
  #findHeight
     ✓ errors on unknown blocks
     ✓ finds height of known blocks
  #findAncestor
     ✓ errors on unknown blocks
     ✓ Finds known ancestors based on on offsets
  #isAncestor
     ✓ returns false if it exceeds the limit
     ✓ finds the ancestor if within the limit
  #ownerAddHeaders
     ✓ appends new links to the chain and fires an event
     ✓ only owner can call it
     ✓ can be called even when the relay is paused
  #ownerAddHeadersWithRetarget
     ✓ appends new links to the chain and fires an event
     ✓ only owner can call it
     ✓ can be called even when the relay is paused

CCBurnRouter
  #ccBurn
     ✓ Reverts since user script length is incorrect
     ✓ Burns teleBTC for user
     ✓ Reverts since user requested amount is zero
     ✓ Reverts since requested amount doesn't cover Bitcoin fee
     ✓ Reverts since allowance is not enough
     ✓ Reverts since locker's locking script is not valid
  #burnProof
     ✓ Submits a valid burn proof (for P2PKH)
     ✓ Reverts since _burnReqIndexes is not sorted
     ✓ Submits a valid burn proof (for P2WPKH)
     ✓ Submits a valid burn proof which doesn't have change vout
     ✓ Reverts since locktime is non-zero
     ✓ Reverts if locking script is not valid
     ✓ Reverts if given indexes doesn't match
     ✓ Reverts since paid fee is not enough
     ✓ Reverts if locker's tx has not been finalized on relay
     ✓ Reverts if vout is null
     ✓ Doesn't accept burn proof since the paid amount is not exact
     ✓ Doesn't accept burn proof since the proof has been submitted before
     ✓ Doesn't accept burn proof since deadline is passed
     ✓ Doesn't accept burn proof since change address is invalid
  #disputeBurn
     ✓ Disputes locker successfully
     ✓ Reverts since locker has been slashed before
     ✓ Reverts since locking script is invalid
     ✓ Reverts since locker has paid before hand
     ✓ Reverts since deadline hasn't reached
  #disputeLocker
     ✓ Dispute the locker who has sent its BTC to external account
     ✓ Reverts since inputs are not valid
     ✓ Reverts since locking script is not valid
     ✓ Reverts since input tx has not finalized
     ✓ Reverts since input tx has been used as burn proof
     ✓ Reverts since outpoint doesn't match with output tx
     ✓ Reverts since tx doesn't belong to locker
     ✓ Reverts since locker may submit input tx as burn proof
  #setters
     ✓ Sets protocol percentage fee
     ✓ Reverts since protocol percentage fee is greater than 10000
     ✓ Sets transfer deadline
     ✓ Fixes transfer deadline
     ✓ can't Fix transfer deadline if finalizationParameter is greater than current transfer deadline
     ✓ Reverts since transfer deadline is smaller than relay finalizatio parameter
     ✓ Reverts since transfer deadline is smaller than relay finalizatio parameter
     ✓ Sets slasher reward
     ✓ Reverts since slasher reward is greater than 100
     ✓ Sets bitcoin fee
     ✓ Sets relay, lockers, teleBTC and treasury
     ✓ Reverts since given address is zero
  #renounce ownership
     ✓ owner can't renounce ownership

CCExchangeRouter
  #ccExchange
     ✓ Exchanges teleBTC for desired exchange token (fixed token = input)
     ✓ Exchanges teleBTC for desired exchange token (fixed token = output)
     ✓ Exchanges teleBTC for desired exchange token through wrapped native token
     ✓ Mints teleBTC since deadline has passed
     ✓ Mints teleBTC since slippage is high (output amount < expected output amount)
     ✓ Mints teleBTC since slippage is high (input amount < required output amount)
     ✓ Mints teleBTC since exchange token doesn't exist
     ✓ Mints teleBTC since exchange token is zero
     ✓ Reverts since given appId doesn't exist
     ✓ Reverts if user hasn't sent BTC to locker
     ✓ Reverts if locker doesn't exist
     ✓ Reverts if the percentage fee is out of range [0,10000)
     ✓ Reverts if the request belongs to another chain
     ✓ Reverts if the request speed is out of range {0,1}
     ✓ Reverts if the request has been used before
     ✓ Reverts since request belongs to an old block header
     ✓ Reverts since lock time is non-zero
     ✓ Reverts if request has not been finalized yet
     ✓ Reverts if paid fee is not sufficient
     ✓ Pays back instant loan (instant cc exchange request)
  #isRequestUsed
     ✓ Checks if the request has been used before (unused)
     ✓ Reverts since the request has been executed before
  #setters
     ✓ Sets protocol percentage fee
     ✓ Reverts since protocol percentage fee is greater than 10000
     ✓ Sets relay, lockers, instant router, teleBTC and treasury
     ✓ Reverts since given address is zero
  #renounce ownership
     ✓ owner can't renounce ownership

CCTransferRouter
  #ccTransfer
     ✓ Mints teleBTC for normal cc transfer request (relay fee is zero)
     ✓ Mints teleBTC for normal cc transfer request (relay fee is non-zero)
     ✓ Mints teleBTC for normal cc transfer request (zero teleporter fee)
     ✓ Mints teleBTC for normal cc transfer request (zero protocol fee)
     ✓ Reverts since request belongs to an old block header
     ✓ Reverts if the request has been used before
     ✓ Reverts if the request has not been finalized on the relay
     ✓ Reverts if the percentage fee is out of range [0,10000)
     ✓ Reverts if chain id is invalid
     ✓ Reverts if app id is invalid
     ✓ Reverts if user sent BTC to invalid locker
     ✓ Reverts if no BTC has been sent to locker
     ✓ Reverts if speed is wrong
     ✓ Reverts if msg.value is lower than relay fee
     ✓ Mints teleBTC for instant cc transfer request
  #isRequestUsed
     ✓ Checks if the request has been used before (unused)
     ✓ Reverts since the request has been executed before
  #setters
     ✓ Sets protocol percentage fee
     ✓ Sets protocol percentage fee
     ✓ Reverts since protocol percentage fee is greater than 10000
     ✓ Sets relay, lockers, instant router, teleBTC and treasury
```

```
      ✓ Reverts since given address is zero

  CollateralPool
    #addCollateral
      ✓ Adds collateral when collateral pool is empty
      ✓ Adds collateral when collateral pool is non-empty
      ✓ Adds collateral after some fee was sent to collateral pool
      ✓ Reverts since user hasn't given allowance to collateral pool
      ✓ Reverts since user address is zero
      ✓ Reverts since amount is zero
    #removeCollateral
      ✓ Removes collateral
      ✓ Removes collateral after some fee was sent to collateral pool
      ✓ Reverts since amount is zero
      ✓ Reverts since balance is not enough
    #equivalentCollateralToken
      ✓ Converts collateral pool token to collateral token
      ✓ Converts collateral pool token to collateral token after transferring some amounts
      ✓ Reverts since liquidity is not enough
      ✓ Reverts since collateral pool is empty
    #equivalentCollateralPoolToken
      ✓ Converts collateral pool token to collateral token
      ✓ Converts collateral pool token to collateral token after transferring some amounts
      ✓ Reverts since liquidity is not enough
      ✓ Reverts since collateral pool is empty
    #setCollateralizationRatio
      ✓ Sets new collateralization ratio
      ✓ Reverts since given ratio is zero
      ✓ Reverts since given ratio is less than 10000
    #renounce ownership
      ✓ owner can't renounce ownership

  CollateralPoolFactory
    #createCollateralPool
      ✓ Creates a collateral pool
      ✓ Reverts since _collateralizationRatio is less than 10000
      ✓ Reverts since collateral pool has been already created
      ✓ Reverts since non-owner account calls the function
      ✓ Reverts since collateral token address is zero
      ✓ Reverts since collateralization ratio is zero
    #removeCollateralPool
      ✓ Removes a collateral pool
      ✓ Reverts since the index is out of range
      ✓ Reverts since the index doesn't belong to collateral token
      ✓ Reverts since the collateral pool doesn't exist
      ✓ Reverts since non-owner account calls the function
    #renounce ownership
      ✓ owner can't renounce ownership

  Instant pool
    #setInstantRouter
      ✓ Non owner accounts can't set instant router
      ✓ Owner can set instant router successfully
    #setInstantPercentageFee
      ✓ Non owner accounts can't set instant router
      ✓ Owner can set instant router successfully
    #setTeleBTC
      ✓ Non owner accounts can't set instant router
      ✓ Owner can set instant router successfully
    #addLiquidity
      ✓ Mints instant pool token when instant pool is empty
      ✓ Mints instant pool token when instant pool is non-empty
      ✓ Mints instant pool token after some amount of teleBTC was transferred directly
      ✓ Mints instant pool token after some amount of teleBTC was added using addLiquidityWithoutMint
      ✓ Reverts since input amount is zero
      ✓ Reverts since user balance is not enough
    #removeLiquidity
      ✓ Burns instant pool token to withdraw teleBTC
      ✓ Burns instant pool token after some amount of teleBTC was transferred directly
      ✓ Burns instant pool token after some amount of teleBTC was added using addLiquidityWithoutMint (before addLiquidity)
      ✓ Burns instant pool token after some amount of teleBTC was added using addLiquidityWithoutMint (after addLiquidity)
      ✓ Reverts since input amount is zero
      ✓ Reverts since user balance is not enough
    #getLoan
      ✓ Gets loan from instant pool
      ✓ Reverts since message sender is not instant router
      ✓ Reverts since available liquidity is not sufficient

  Instant Router
    #instantCCTransfer
      ✓ Gives instant loan to user
      ✓ Reverts instant transfer since contract is paused
      ✓ Check unpause for instant transfer
      ✓ Reverts since deadline has paased
      ✓ Reverts since collateral is not acceptable
      ✓ Reverts since instant pool liquidity is not enough
      ✓ Reverts because has reached to max loan number
    #instantCCExchange
      ✓ Gives loan to user and exchanges teleBTC to output token
      ✓ Reverts instant exchange since contract is paused
      ✓ Check unpause in instant exchange
      ✓ Reverts since deadline has paased
      ✓ Reverts since path is invalid
      ✓ Reverts since instant pool liquidity is not enough
      ✓ Reverts since collateral is not acceptable
      ✓ Reverts since swap was not successful
    #payBackLoan
      ✓ Paybacks a debt when user has one unpaid debt
      ✓ Paybacks a debt when user has two unpaid debts
      ✓ Paybacks a debt and sends remained amount to user when user has two unpaid debts
      ✓ Paybacks debts when user has two unpaid debts
      ✓ Sends teleBTC back to user since payback amount is not enough
      ✓ Sends teleBTC back to user since deadline has passed
    #slashUser
      ✓ Slash user reverted because big gap between dex and oracle
      ✓ Slashes user and pays instant loan fully
      ✓ Slashes user and pays instant loan partially
      ✓ Slashes user and pays instant loan partially (amount from oracle is bigger)
      ✓ Slashes user and pays instant loan partially (high swap result)
      ✓ Reverts since request index is out of range
      ✓ Reverts since payback deadline has not passed yet
      ✓ Reverts since there's a big gap between price oracle and dex
      ✓ Reverts since liquidity pool doesn't exist
    #setters
      ✓ Sets slasher percentage reward
      ✓ Reverts since slasher percentage reward is greater than 100
      ✓ Sets payback deadline
      ✓ Fixes payback deadline
      ✓ can't Fix payback deadline if finalizationParameter is greater than current payback deadline
      ✓ Reverts since payback deadline is lower than relay finalization parameter
      ✓ Sets relay, lockers, instant router, teleBTC and treasury
      ✓ Reverts since given address is zero
      ✓ Reverted because non-owner account is calling

  Lockers
    #initialize
      ✓ initialize can be called only once
      ✓ initialize cant be called with zero address
      ✓ initialize cant be called with zero amount
      ✓ initialize cant be called LR greater than CR
      ✓ initialize cant be called with Price discount greater than 100%
    #addMinter
      ✓ can't add zero address as minter
      ✓ only owner can add a minter
      ✓ owner successfully adds a minter
      ✓ can't add an account that already is minter
    #removeMinter
      ✓ can't remove zero address as minter
```

```
    ✓ only owner can add a minter
    ✓ owner can't remove an account from minter that it's not minter ATM
    ✓ owner successfully removes an account from minters
  #addBurner
    ✓ can't add zero address as burner
    ✓ only owner can add a burner
    ✓ owner successfully adds a burner
    ✓ can't add an account that already is burner
  #removeBurner
    ✓ can't remove zero address as burner
    ✓ only owner can add a burner
    ✓ owner can't remove an account from burners that it's not burner ATM
    ✓ owner successfully removes an account from burner
  #pauseLocker
    ✓ only admin can pause locker
    ✓ contract paused successsfully
    ✓ can't pause when already paused
  #unPauseLocker
    ✓ only admin can un-pause locker
    ✓ can't un-pause when already un-paused
    ✓ contract un-paused successsfully
  #setTeleportDAOToken
    ✓ non owners can't call setTeleportDAOToken
    ✓ only owner can call setTeleportDAOToken
  #setLockerPercentageFee
    ✓ non owners can't call setLockerPercentageFee
    ✓ only owner can call setLockerPercentageFee
  #setPriceWithDiscountRatio
    ✓ non owners can't call setPriceWithDiscountRatio
    ✓ only owner can call setPriceWithDiscountRatio
  #setMinRequiredTDTLockedAmount
    ✓ non owners can't call setMinRequiredTDTLockedAmount
    ✓ only owner can call setMinRequiredTDTLockedAmount
  #setMinRequiredTNTLockedAmount
    ✓ non owners can't call setMinRequiredTNTLockedAmount
    ✓ only owner can call setMinRequiredTNTLockedAmount
  #setPriceOracle
    ✓ price oracle can't be zero address
    ✓ non owners can't call setPriceOracle
    ✓ only owner can call setPriceOracle
  #setCCBurnRouter
    ✓ cc burn router can't be zero address
    ✓ non owners can't call setCCBurnRouter
    ✓ only owner can call setCCBurnRouter
  #setExchangeConnector
    ✓ exchange connector can't be zero address
    ✓ non owners can't call setExchangeConnector
    ✓ only owner can call setExchangeConnector
  #setTeleBTC
    ✓ tele BTC can't be zero address
    ✓ non owners can't call setTeleBTC
    ✓ only owner can call setTeleBTC
  #setCollateralRatio
    ✓ non owners can't call setCollateralRatio
    ✓ only owner can call setCollateralRatio
  #setLiquidationRatio
    ✓ non owners can't call setLiquidationRatio
    ✓ only owner can call setLiquidationRatio
  #requestToBecomeLocker
    ✓ setting low TeleportDao token
    ✓ not approving TeleportDao token
    ✓ low message value
    ✓ successful request to become locker
    ✓ a locker can't requestToBecomeLocker twice
    ✓ a redeem script hash can't be used twice
  #revokeRequest
    ✓ trying to revoke a non existing request
    ✓ successful revoke
  #addLocker
    ✓ trying to add a non existing request as a locker
    ✓ adding a locker
  #requestInactivation
    ✓ trying to request to remove a non existing locker
    ✓ successfully request to be removed
  #requestActivation
    ✓ trying to activate a non existing locker
    ✓ successfully request to be activated
  #selfRemoveLocker
    ✓ a non-existing locker can't be removed
    ✓ can't remove a locker if it doesn't request to be removed
    ✓ the locker can't be removed because netMinted is not zero
    ✓ the locker is removed successfully
  #slashIdleLocker
    ✓ only cc burn can call slash locker function
    ✓ slash locker reverts when the target address is not locker
    ✓ can't slash more than collateral
    ✓ cc burn can slash a locker
  #slashTheifLocker
    ✓ only cc burn can call slash locker function
    ✓ slash locker reverts when the target address is not locker
    ✓ cc burn can slash a locker
  #buySlashedCollateralOfLocker
    ✓ reverts when the target address is not locker
    ✓ not enough slashed amount to buy
    ✓ can't slash because needed BTC is more than existing
    ✓ can buy slashing amount
  #mint
    ✓ Mints tele BTC
    ✓ can't mint tele BTC above capacity
  #burn
    ✓ Burns tele BTC
  #liquidateLocker
    ✓ liquidate locker reverts when the target address is not locker
    ✓ can't liquidate because it's above liquidation ratio
    ✓ can't liquidate because it's above the liquidated amount
    ✓ successfully liquidate the locker
  #addCollateral
    ✓ can't add collateral for a non locker account
    ✓ reverts because of insufficient msg value
    ✓ adding collateral to the locker
  #priceOfOneUnitOfCollateralInBTC
    ✓ return what price oracle returned
  #mint
    ✓ only owner can call renounceOwnership
    ✓ can't mint because receipt is zero address
    ✓ can't mint since locker is inactive
  #removeCollateral
    ✓ can't remove collateral for a non locker account
    ✓ reverts because it's more than capacity
    ✓ reverts because it's more than capacity
    ✓ reverts because it becomes below the min required collateral
    ✓ remove collateral successfully

PriceOracle
  #addExchangeConnector
    ✓ Adds an exchange router
    ✓ Reverts since exchange router already exists
  #removeExchangeConnector
    ✓ Removes an exchange router
    ✓ Reverts since exchange router doesn't exist
  #setPriceProxy
    ✓ Sets a price proxy
    ✓ Removes a price proxy
    ✓ Reverts since one of tokens is zero
  #equivalentOutputAmountFromOracle
    ✓ Gets equal amount of output token when TT/ATT proxy has been set
    ✓ Gets equal amount of output token when ATT/TT proxy has been set
    ✓ Gets equal amount of output token when input token is native token
```

```
    ✓ Gets equal amount of output token when output token is native token
    ✓ Gets equal amount of output token when price decimal is zero
    ✓ Gets equal amount of output token when all decimals are zero
    ✓ Reverts since one of the tokens is zero
    ✓ Reverts since returned price is zero
    ✓ Reverts since one of the tokens doesn't exist
  #equivalentOutputAmountFromExchange
    ✓ Gets equal amount of output token
    ✓ Gets equal amount of output token when input token is native token
    ✓ Gets equal amount of output token when output token is native token
    ✓ Reverts since one of the tokens is zero
    ✓ Reverts since pair does not exist in exchange
  #equivalentOutputAmount
    ✓ Gets equal amount of output token when delay is not acceptable, but no other exchange exists (only oracle)
    ✓ Gets equal amount of output token when delay is not acceptable, but exchange does not have the pair (only oracle)
    ✓ Gets equal amount of output token when delay is acceptable (only oracle)
    ✓ Gets equal amount of output token when delay is acceptable (oracle and router)
    ✓ Gets equal amount of output token when delay is not acceptable (oracle and router)
    ✓ Gets equal amount of output token when delay is not acceptable and input token is native token (oracle and router)
    ✓ Gets equal amount of output token when delay is acceptable and input token is native token (oracle and router)
    ✓ Gets equal amount of output token when delay is not acceptable and output token is native token (oracle and router)
    ✓ Gets equal amount of output token when delay is acceptable and output token is native token (oracle and router)
    ✓ Gets equal amount of output token when delay is not acceptable and output token is native token (oracle and router)
    ✓ Gets equal amount of output token when price proxy doesn't exist (only router)
    ✓ Gets equal amount of output token when delay is acceptable, but no other exchange exists (only oracle)
    ✓ Gets equal amount of output token when delay is acceptable, but no other exchange exists (only oracle)
    ✓ Gets equal amount of output token when delay is acceptable, but exchange does not have the pair (only oracle)
    ✓ Gets equal amount of output token when delay is acceptable, but exchange does not have the pair (only oracle)
    ✓ Reverts since no price feed was found (no oracle no router)
    ✓ Reverts since no price feed was found (no oracle)
  #setters
    ✓ Sets acceptable delay
    ✓ Sets oracle native token
    ✓ Reverts since given address is zero
    ✓ renounceOwnership

TeleBTC
  #mint rate limit
    ✓ can't mint more than maximum mint limit in one transaction
    ✓ can't mint more than maximum mint limit in one epoch
    ✓ after an epoch, mint rate limit will be reset
  #burn and mint
    ✓ non burner account can't burn tokens
    ✓ non minter account can't mint tokens
    ✓ minters can mint tokens and burner can burn tokens
  #minter
    ✓ add minter
    ✓ can't add zero address as minter
    ✓ can't add minter twice
    ✓ remove minter
  #burner
    ✓ add burner
    ✓ can't add zero address as burner
    ✓ can't add burner twice
    ✓ remove burner
  Renounce ownership
    ✓ owner can't renounce his ownership
  Setters
    ✓ none owner accounts can't change maximum mint limit
    ✓ owner account can change maximum mint limit
    ✓ none owner accounts can't change epoch length
    ✓ can't change epoch length to zero
    ✓ owner account can change epoch length
    ✓ can't change epoch length to zero
  Getters
    ✓ decimal is correct

UniswapV2Connector
  #getInputAmount
    ✓ Finds needed input amount
    ✓ Returns true when there is an indirect path
    ✓ Returns false when there is no even an indirect path
    ✓ Returns false since liquidity pool does not exist
    ✓ Returns false since output amount is greater than output reserve
    ✓ Reverts since one of token's addresses is zero
  #getOutputAmount
    ✓ Finds output amount
    ✓ Returns false since liquidity pool does not exist
    ✓ Returns true when there is indirect path
    ✓ Returns false when there is no evenn an indirect path
    ✓ Reverts since one of token's addresses is zero
  #swap
    ✓ Swaps indirect path fails
    ✓ Swaps indirect path
    ✓ Swaps fixed non-WETH for non-WETH
    ✓ Swaps non-WETH for fixed non-WETH
    ✓ Swaps fixed non-WETH for WETH
    ✓ Swaps non-WETH for fixed WETH
    ✓ Should not exchange since expected output amount is high
    ✓ Should not exchange since input amount is not enough
    ✓ Should not exchange since deadline has passed
    ✓ Should not exchange since liquidity pool doesn't exist
    ✓ Should not exchange since path only has one element
  #isPathValid
    ✓ Returns true since path is valid
    ✓ Returns false since path is empty
    ✓ Returns false since path only has one element
    ✓ Returns false since liquidity pool doesn't exist
    ✓ Returns false since path is invalid
  #setters
    ✓ Sets new exchange router
    ✓ Reverts since exchange router address is zero
    ✓ Reverts since exchange router address is invalid
    ✓ Sets liquidity pool factory and wrapped native token
```

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
3b1836da20f040a9d33d4868a78c0872cf00a6fc06f0c6d3ab76a7e5656b224e ./contracts/uniswap/v2-periphery/UniswapV2Migrator.sol

fd917d76dc9196d71cba1c3ae131828a7304147e5921a4624132cce2478d1dbe ./contracts/uniswap/v2-periphery/UniswapV2Router01.sol

61a9b5bd12501511a030ec89e2e27d4c492e6a649dddc6d91f32a0a759fd0e0d ./contracts/uniswap/v2-periphery/UniswapV2Router02.sol

2bde0b8be2c4b14601d153c483cd1db088c64eacce8c22c574f3efcb66513567 ./contracts/uniswap/v2-periphery/libraries/SafeMath.sol

369a92ec54d78eb988b726e3a8d814267806d707bbe1aa7c1dff49d295279a80 ./contracts/uniswap/v2-periphery/libraries/TransferHelper.sol

f602a98464f52e59c29d6e6acb84bb0b03725acf4df864a0333081de00090ed4 ./contracts/uniswap/v2-periphery/libraries/UniswapV2Library.sol
```

534161476e433dc584d085c2192e1977c4e8f17d3ee6f2c84bfd557df9dbf610 ./contracts/uniswap/v2-periphery/interfaces/IERC20.sol

a6c52b253c720d1c8fbbcb60f437d89c1e9bb41a8ebe638814eb25879ad3c8d8 ./contracts/uniswap/v2-periphery/interfaces/IUniswapV2Migrator.sol

9e9232b0ab8af12bf698a622047a0057ab2b5b068360e24c8599576a40653601 ./contracts/uniswap/v2-periphery/interfaces/IUniswapV2Router01.sol

add2f9ec336a24dfe0fcf25cd27fd11860fa09f8e303867f5188b2b1769b31e4 ./contracts/uniswap/v2-periphery/interfaces/IUniswapV2Router02.sol

d1e9b249dc6368fc02dcd4b6c27d6c29393e84763ab77cbaa3adf77c0e078b9b ./contracts/uniswap/v2-periphery/interfaces/IWETH.sol

092986031594386fe1dbc05ccffb559ca338cc92c21d1fd630c9a617a647f9eb ./contracts/uniswap/v2-periphery/interfaces/V1/IUniswapV1Exchange.sol

468a6fa18b3d0e651d87e9704c1807fc671505ec539da3fb370321897402bb7b ./contracts/uniswap/v2-periphery/interfaces/V1/IUniswapV1Factory.sol

7951af23649481c07faa50185dc09f6cc2c2a106ebbb940611bbab3f5cd4a875 ./contracts/uniswap/v2-core/UniswapV2ERC20.sol

190e8295d8f6ca088ac7d109f91ad2b4dff326f6fbab28b3fd64c2a9d1e50603 ./contracts/uniswap/v2-core/UniswapV2Factory.sol

65a56226748b15167885246f3384288afa7745a34708b1e6ad1acd04764e714f ./contracts/uniswap/v2-core/UniswapV2Pair.sol

e4a9d451964a0689be2b244322a353de143ca4248d8736d91aca4ffadca4325f ./contracts/uniswap/v2-core/libraries/Math.sol

4b1c95ff75de7342e0fadff58064820a4eb7c2fcb422a75b4994980ce8e216ae ./contracts/uniswap/v2-core/libraries/SafeMath.sol

6633b57b0723b1d72e08cc3e8b29f0af838294e59863b6cdcce95a141ed02cdb ./contracts/uniswap/v2-core/libraries/UQ112x112.sol

534161476e433dc584d085c2192e1977c4e8f17d3ee6f2c84bfd557df9dbf610 ./contracts/uniswap/v2-core/interfaces/IERC20.sol

17613dd95f744dfb114667190beee15f02562f59e76f099cefcce04c75cd2a52 ./contracts/uniswap/v2-core/interfaces/IUniswapV2Callee.sol

b05399bb92c56bdb470484b1a5e8081cc652bfae08f66d93fdfd7b9f1591c466 ./contracts/uniswap/v2-core/interfaces/IUniswapV2ERC20.sol

3dd4c1f051cee242d1c81b3868d19d983706f47dc6d4e61c83e8645dab7b190f ./contracts/uniswap/v2-core/interfaces/IUniswapV2Factory.sol

d031a0cf0541e16cc08a0772453796dcbf77727976822ac038dbea47e16171cb ./contracts/uniswap/v2-core/interfaces/IUniswapV2Pair.sol

16c9200c19952d308cd8095613dd52cd4169fb0c9c11d179c07b8514ddeaba5c ./contracts/types/DataTypes.sol

30d2c8fac55c2e1679e4ad4431876aabd081eb27f3640be66a56fbfc471d26b6 ./contracts/types/ScriptTypesEnum.sol

6709b8f72b7a661c0f1c6e5e5eaef7af4e2a3d03706ab92f9219791b87a3c85f ./contracts/routers/CCBurnRouter.sol

c4eca1e13101de74e051b97ecdf4f4a41b6e2faeb3e487535226446bf688ca6f ./contracts/routers/CCExchangeRouter.sol

871a73ca46e2e6698a0a6f02646dbbb2223f6909e0a179a06d01f45bae30fec4 ./contracts/routers/CCTransferRouter.sol

ae64b532ed753bb39859c9f3d5782c4c06eda8411977f1282e491f37aa535aee ./contracts/routers/InstantRouter.sol

060bfce0f9577f4e22807f9c69458140ccc3c5b186d8867d3e8330f449dd5823 ./contracts/routers/interfaces/ICCBurnRouter.sol

dc72bf31829a59a8012f21ad1154848753e35a2dc72a11bc32742e103feebc80 ./contracts/routers/interfaces/ICCExchangeRouter.sol

1090f2ade94c56e7fc08b5da7bdc316af5dd4dcb9314f858c375efae5c1a555f ./contracts/routers/interfaces/ICCTransferRouter.sol

d65ef848101e4190a031c6aa1bf4ae8ad8f4c6ff9b7ea1c1aa844c300b0261d5 ./contracts/routers/interfaces/IInstantRouter.sol

da1ce13a70cd2da695c1f68a8ec1ede03c055498a66a7ed3a148c98ee58f4f3d ./contracts/relay/BitcoinRelay.sol

a117cbb8809c02040a5733384a6d2332fc1ae18cc502db5f9a69faabd08e6ac7 ./contracts/relay/interfaces/IBitcoinRelay.sol

ec5854201fa27477aedf910850962c3d82264f67890292e6c0c423141929d40d ./contracts/pools/CollateralPool.sol

3c3a975bf3f2bd615610c90bea3b909df6b67d45cf7520ed0ca9c4982d9ddb80 ./contracts/pools/CollateralPoolFactory.sol

0a7547eb34e180e4c381aa86c32867a7359d715fa48a7015da3a1b150caa75dd ./contracts/pools/InstantPool.sol

59a38ca746014a49513ffa10360c2f39185ab34463720f8977cf373bf945104e ./contracts/pools/interfaces/ICollateralPool.sol

2855cf40c54e4814e3e68218dfb04ca6146ff57902b6d76fd5a3f2af12e0c6f7 ./contracts/pools/interfaces/ICollateralPoolFactory.sol

af8058e85e03a1bd28bfce8623660bb44782cf5c662b80decb65fce388a14a1d ./contracts/pools/interfaces/IInstantPool.sol

79931fe0671fb599d456efba3be8138606b990268304bc86000cbd1f23a96019 ./contracts/oracle/PriceOracle.sol

d2bcd04ed6d86aed871e997844aa41e97f57d1c6f519023a330ce008ea9e9592 ./contracts/oracle/interfaces/IPriceOracle.sol

a96752966ae058aa9fea329f8cc9f417512f6788e3cbe2a078f9d8403130e465 ./contracts/lockers/LockersLogic.sol

83bedfcdd41ade5209c0f3839158dd212bea6479ec1210e66ba6d59994a241c4 ./contracts/lockers/LockersProxy.sol

33ae737a8b16a413072e7bdea7d3ff760d9dd35dcc5b64a41056099e63378a92 ./contracts/lockers/interfaces/ILockers.sol

8db9b02da82fa6f9a28c2fd7de4f8bad905ddaef62fbbf608ae1424bf6e2c2d9 ./contracts/libraries/BitcoinHelper.sol

2d7ce61b42fac4445c4664e51d03dd39025dc3f09dab8b38cf500b2018d3d7b7 ./contracts/libraries/LockersLib.sol

436eccc655e287c4eff5c336f5b76e11a6020ca40816d7d75402f07459524a3a ./contracts/libraries/RequestHelper.sol

7abc21d6352f754d73e6d7ff0d66a0d6f17f976b92313aeec9279c5aacdf14bb ./contracts/libraries/SafeMath.sol

7ae4a9572e95a736cc809e2de4d6b283729bfd538023de5f3bdffd001fc2830a ./contracts/libraries/TypedMemView.sol

eb0167b1c14cef3031e76e798268da52fd19d43c30331f502f95bc5d5ad252f3 ./contracts/erc20/Context.sol

9204b374a9e070d6e0394892635a00855dd0872ebf969754632ee614cdcb8ab4 ./contracts/erc20/ERC20.sol

ff09e1d1adc228fce4710aba72186455a8b2b1d8f0084b6f045b8342cf4a8329 ./contracts/erc20/ERC20AsDot.sol

481572e7e7d07f8450b3cf09ad287539d75588978cc92d974279f66dac7700f8 ./contracts/erc20/ERC20AsLink.sol

ebc2200abece0ab4c9251deddcae581927d08dd98fc1aef2e00f8e90c5c63c20 ./contracts/erc20/TeleBTC.sol

1913d9e310d23202e9b986ac8ac5ce84dc341a56afaff66a4d362dd6f6e342b1 ./contracts/erc20/WETH.sol

056bda3ccf430286b96e784f655f8bbfa6656cce53eb84841e24c52684447615 ./contracts/erc20/interfaces/IERC20.sol

28f556455674edea07a89cd7a6791ca4488ad847b5b4b16cbf0cd6f0b0a10d22 ./contracts/erc20/interfaces/ITeleBTC.sol

381f2d5764596e6e8cac5c79462d084455b269ea57359f32328874db7b19fe7a ./contracts/erc20/interfaces/IWETH.sol

d0d30063accfbfc56704ce3f6afaf375e634b1bf629681dafbd2993b99407ecd ./contracts/connectors/UniswapV2Connector.sol

4fc3e097c4caeb5f9159f8a0fa050ad4c80683a5e00a8fab9ff55491913667d4 ./contracts/connectors/interfaces/IExchangeConnector.sol

Tests

```
3c5c72927ff7342dcb5351f9557cba4e10f837fc5b05cfb17a3a6725242f5e47 ./test/bitcoin_relay.test.ts
df631a27091aea0d3de6ec3ebde4d8579b9e8efe742fb0c2912ad174becb25a1 ./test/block_utils.js
6fe70957fb4e3544a44ef5d86c58a465cfd5335d4102e779807acd66374f13d6 ./test/block_utils.ts
c4ececbf69d6791e96624919287e743f4f90d4aa8b848a4293e2ce53e02603b5 ./test/cc_burn_router.test.ts
5303cfe1bcdd827ddec8fe75369ac7278271e75be14ceb87332bb59b4559ccc6 ./test/cc_exchange_router.test.ts
a228f8d2b3c0208a7dc884f69febf65fac29a02bd76201d9a9d6ffae8a62a0c6 ./test/cc_transfer_router.test.ts
c24c618bab4953e2b18b203ca9dffe261a5a601f5a08b5992e8a6106623535f7 ./test/collateral_pool.test.ts
447ce91e41442977864872238a48fef7114f5ac0e4076190aeef03a3ea797427 ./test/collateral_pool_factory.test.ts
fd647595691c0b8c5f5317d5905ebec06d7768b39f4438299d606cf64d9e9dd6 ./test/instant_pool.test.ts
6d98e9b209d1ab0ce35c0f0448223276e97d7ce3ef1a4418bc2bc1e9c7722ea3 ./test/instant_router.test.ts
43f94b875f23e40e58326a0572bbd294f86ea85a61b01a852911bf3a2ff4a8dd ./test/lockers.test.ts
ad74757ef49f8aa3856ecf03ff163edca5fe9987cd8cc90d589ab499c8a9581c ./test/price_oracle.test.ts
d726dba8107f8cc6d35d78760a4f0e5764fede241420963776fd782acfba1c06 ./test/uniswap_v2_connector.test.ts
eddbd143da5dacf0e6ba55257dc45a8fb9b1d18f273408c1c4bfbdbc1b35c427 ./test/utils.js
```

Changelog

- 2022-10-17 - Initial report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.