Atomic Loans

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type: Cross-chain Lending Platform
Auditors: Ed Zulkoski, Senior Security Engineer
          Sebastian Banescu, Senior Research Engineer
          Kacper Bąk, Senior Research Engineer
Timeline: 2020-01-20 through 2020-04-29
EVM: Istanbul
Languages: Solidity, Javascript, Bitcoin Script
Specification: Atomic Loans Wiki

Atomic Loans: Cryptocurrency Debt Instruments

Source Code

<table>
<thead>
<tr>
<th>Repository</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>atomicloans-eth-contracts</td>
<td>10c2493</td>
</tr>
<tr>
<td>atomicloans-oracle-contracts</td>
<td>06c90e0</td>
</tr>
<tr>
<td>chainabstractionlayer-loans</td>
<td>98ea474</td>
</tr>
</tbody>
</table>

Goals

• Do the smart contracts and Bitcoin scripts correctly implement the atomic loan protocol?
• Can funds be locked or stolen by adversaries at any point during the protocol?
• Does the protocol correctly interact with oracle protocols?

Changelog

• 2020-01-31 - Initial report
• 2020-02-21 - Revised report based on commit 8016c19, 39463fe, and 67d3df1
• 2020-03-31 - Revised report based on commit 8016c19
• 2020-04-29 - Revised report based on commit 878917d

Total Issues: 11 (11 Resolved)
High Risk Issues: 0 (0 Resolved)
Medium Risk Issues: 0 (0 Resolved)
Low Risk Issues: 2 (2 Resolved)
Informational Risk Issues: 7 (7 Resolved)
Undetermined Risk Issues: 2 (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).

Resolved
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

Quantstamp has assessed the Atomic Loans smart contracts and Bitcoin scripts, and consider them to be well-architected and adherent to the provided specification. No critical security issues were detected during this audit, however we provide several suggestions for code improvements based on issues found during the audit. We recommend these issues be reviewed and resolved prior to the code being used in production.

Disclaimer: This audit only assessed a subset of the code contained in the above repositories. Specifically, it is scoped to the following code:

- In `atomicloans-eth-contracts`, the contracts `Funds.sol`, `Loans.sol`, and `Sales.sol`;
- In `atomicloans-oracle-contracts`, the contracts `Medianizer.sol`, `Oracle.sol`, `chainLink/*`, and `oracle/*`;
- In `chainabstractionlayer-loans`, the Bitcoin scripts in `BitcoinCollateralProvider.js` and `BitcoinCollateralSwapProvider.js`.

Update: The Atomic Loans team has addressed our concerns as of commit `8016c19` of `atomicloans-eth-contracts`, `3f963fe` of `atomicloans-oracle-contracts`, and `67d3df1` of `chainabstractionlayer-loans`. We commend the Atomic Loans team’s pro-active and well-organized approach to addressing all findings (including best practices), which significantly streamlined the re-audit process.

Update 2: The Atomic Loans team has addressed our comments as of commit `8016c19`. Only one informational issue was found.

Update 3: The Atomic Loans team has addressed our comments as of commit `878917d`. Only one informational issue was found.

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Severity</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>QSP-1</td>
<td>Testing code included amongst main code repository</td>
<td>vLow</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-2</td>
<td>Loans can be requested for extremely long periods of time</td>
<td>vLow</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-3</td>
<td>Code does not adhere to the checks-effects-interactions pattern</td>
<td>Informational</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-4</td>
<td>Unlocked Pragma</td>
<td>Informational</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-5</td>
<td>Centralization of Power</td>
<td>Informational</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-6</td>
<td>Undocumented magic constants</td>
<td>Informational</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-7</td>
<td>Integer Overflow / Underflow</td>
<td>Informational</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-8</td>
<td>Downcasting and Upcasting may lead to unexpected results</td>
<td>Informational</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-9</td>
<td>Missing input validation</td>
<td>Informational</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-10</td>
<td>Gas Usage / For Loop Concerns</td>
<td>? Undetermined</td>
<td>Resolved</td>
</tr>
<tr>
<td>QSP-11</td>
<td>Possible unhandled exception in <code>BitcoinCollateralProvider.setPaymentVariants()</code></td>
<td>? Undetermined</td>
<td>Resolved</td>
</tr>
</tbody>
</table>
Quantstamp’s objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

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:

- Truffle
- Ganache
- SolidityCoverage
- Mythril
- Truffle-Flattener
- Slither
- bitcoind
Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
4. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
5. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
6. Installed the Mythril tool from Pypi: `pip3 install mythril`
7. Ran the Mythril tool on each contract: `myth -x path/to/contract`
8. Installed the Slither tool: `pip install slither-analyzer`
9. Run Slither from the project directory: `slither .`

Assessment

Findings

QSP-1 Testing code included amongst main code repository

Severity: Low Risk
Status: Resolved
File(s) affected: BlockchainInfo.sol, Funds.sol, ChainLink.sol, ChainlinkedTesting.sol, OraclizeAPITesting.sol, CoinMarketCap.sol, CryptoCompare.sol, Gimini.sol, SoChain.sol, Oraclize.sol

Description: There are several locations where mainnet constants are commented out in favor of testnet constants:

1. On L7,9 of BlockchainInfo.sol, there are commented out job IDs for mainnet.
2. It should be checked that the job IDs defined on L6-10 of BlockchainInfo.sol are up-to-date. We could only find f291f8597d174f4aa1983b0e27ae160f on either of these pages: https://docs.chain.link/docs/decentralized-oracles-ethereum-mainnet and https://docs.chain.link/docs/testnet-oracles.
3. In Funds.sol, on L36/37 and L403/419, an exception is made for the deployer during testing. These should be removed for production.
4. On L3 of ChainLink.sol, a mainnet import is commented out in favor of a testing contract on L4. A similar issue exists in Oraclize.sol.
5. In the files CoinMarketCap.sol, CryptoCompare.sol, Gimini.sol and SoChain.sol, there are mainnet constants commented out at the top of each contract.

Further, there are two testing contracts: ChainlinkedTesting.sol and OraclizeAPITesting.sol included amongst the production code.

Recommendation: Ensure that the correct constants are used in the contracts before deployment to mainnet. Remove testing contracts from the main code directories in favor of a test-directory.

QSP-2 Loans can be requested for extremely long periods of time

Severity: Low Risk
Status: Resolved
File(s) affected: Funds.sol

Description: There are 3 functions (create(), createCustom() and update()) that employ the following `require` statement to prevent requesting a loan for "eternity" according to the code comment at the end of the line of code: `require(ensureNotZero(maxLoanDur_) != 2**256-1 || ensureNotZero(fundExpiry_) != 2**256-1); // Make sure someone can't request a loan for eternity. Even though this require statement prevents someone from setting both maxLoanDur_ and fundExpiry_ to the maximum value at the same time, one of these variables can still be set to the max because of the OR (||) inside the require condition. Moreover, this condition would be true even if one of the 2 variables are equal to 2**256-2, which are still very large values for any practical purposes.

Recommendation: Replace `||` with `&&`. Consider replacing 2**256-1 with a practical value that indicates the maximum duration of a loan, e.g., 30 years or something reasonable.

QSP-3 Code does not adhere to the checks-effects-interactions pattern

Severity: Informational
Status: Resolved
File(s) affected: Funds.sol, Loans.sol

Description: In order to avoid any chance of reentrancy issues, it is generally advised that contracts adopt the checks-effects-interactions pattern. This involves first checking the blockchain state and transaction data to determine validity of the transaction, then performing any state variable update effects, and finally invoking any external contract interactions. This ensures that the state of the contract has been fully updated before interacting with any (potentially malicious) external code.

Recommendation: Although these token contracts are typically safe and trusted by all parties in a loan, we recommend updating these functions to adhere to the checks-effects-interactions pattern, by moving external function calls later in the functions where possible.
QSP-4 Unlocked Pragma
Severity: Informational
Status: Resolved
File(s) affected: All Contracts

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. 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 behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-5 Centralization of Power
Severity: Informational
Status: Resolved
File(s) affected: Loans.sol, Funds.sol

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, this centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

Although many of the centralized components of the contracts are properly documented, several require additional documentation:

1. The setter functions `Loans.setSales()` and `Loans.setP2WSH()` may update the associated `sales` and `p2wsh` contracts exactly one time. Although this mitigates centralized power, extra documentation should be added to indicate the design decisions.
2. Similarly, the documentation for `Funds.setLoans()` and `Funds.setCompound()` should be updated to describe the centralized aspects of the functions.
3. The setter function `Loans.setOnDemandSpv()` may update the associated `onDemandSpv` contract at any time. Regarding the TODO and question on L372, if this address gets updated, the owner could potentially produce false transaction data from the bitcoin network which could affect existing loans. Therefore, we recommend making this field non-upgradeable, and favor re-deploying the contract if the address needs to be updated.

4. `Medianizer` is inherently centralized and based on the addresses by the deployer for the oracles.

Recommendation: Users should be made aware of the roles and responsibilities of the deployer in all contracts. Where appropriate, the amount of centralized privileges should be limited as much as possible.

QSP-6 Undocumented magic constants
Severity: Informational
Status: Resolved
File(s) affected: ChainLink.sol, Loans.sol, Funds.sol, Sales.sol, BlockchainInfo.sol, CoinMarketCap.sol, CryptoCompare.sol, Gemini.sol, SoChain.sol

Description: There are several constants used in the code that do not have associated documentation:

1. In `Loans.sol` on L315, what does `10**12` represent?
2. In `Loans.sol` on L669, there is a constant 3.
3. In `Sales.sol` on L278, there is a constant 3.
4. In `Funds.sol` on L21, L375, L420, L480, and L711 the value `2**256 - 1` is used, which is equal to the `DEFAULT_MAX_LOAN_AMT` constant value defined in the same contract.
5. In `Funds.sol` on L13, it is not clear if `1000000000957505470807876289` represents 3%.
6. In `Funds.sol` on L104, it is not clear if `1000000000236936836262880196` represents 0.75%.
7. In `Funds.sol` on L51, it is not clear if `10000000001552292157372869` represents ~1%.
8. In `ChainLink.sol` on L23, why is `2 * LINK` tokens used? Similarly, on L51, the constant `43200` is used.
9. In `BlockchainInfo.sol`, on L20 and L30 the value `100000000000000000000000` is used. Similarly, L21 and L31 use the value 2.
10. In `CoinMarketCap.sol`, on L25 and L40 the value `10000000000000000000000000000000` is used. Similarly, L26 and L41 use the value 2, and L18 and L33 use the value 5.
11. In `CryptoCompare.sol`, on L20 and L30 the value `10000000000000000000000000000000` is used. Similarly, L21 and L31 use the value 2.
12. In `Gemini.sol`, L20 and L30 use the value `10000000000000000000000000000000`, and L21 and L31 use the value 2.
13. In `SoChain.sol`, L20 and L30 use the value `10000000000000000000000000000000`, and L21 and L31 use the value 2.

Recommendation: Ensure that all constants are as intended, and use named constants where appropriate. Add documentation explaining the rationale behind each constant.
QSP-7 Integer Overflow / Underflow
Severity: Informational
Status: Resolved
File(s) affected: Funds.sol, BitcoinCollateralProvider.js

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute. Integer overflow and underflow may cause many unexpected kinds of behavior and was the core reason for the batchOverflow attack. Here's an example with uint8 variables, meaning unsigned integers with a range of 0..255.

```solidity
function under_over_flow() public {
    uint8 num_players = 0;
    num_players = num_players - 1;  // 0 - 1 now equals 255!
    if (num_players == 255) {
        emit LogUnderflow();        // underflow occurred
    }
    uint8 jackpot = 255;
    jackpot = jackpot + 1;          // 255 + 1 now equals 0!
    if (jackpot == 0) {
        emit LogOverflow();         // overflow occurred
    }
}
```

There is a potential integer overflow on L677 in the expression `now > (lastGlobalInterestUpdated + interestUpdateDelay)`, because the deployer is allowed to set the `interestUpdateDelay` to any value. This would have the effect of updating the global interest value in a scenario where it should not be updated. In BitcoinCollateralProvider.js, L476-477 are used to convert 2 values to BigNumber. These values are later added together using the `+` sign (on L506), instead of the `plus` function of BigNumber, which could lead to an overflow or unexpected results.

Recommendation: Use add from SafeMath in Funds.sol. For BitcoinCollateralProvider.js, use the built-in functions of BigNumber to avoid unexpected results when processing large numbers.

QSP-8 Downcasting and Upcasting may lead to unexpected results
Severity: Informational
Status: Resolved
File(s) affected: Medianizer.sol, DSMath.sol

Description: There is an explicit downcast of `amount` from `uint256` to `uint128` on L63 of Medianizer.sol. This downcast is performed inside the `div` function defined on L21 of DSMath.sol with formal parameters of type `uint256`. This downcasting and upcasting may lead to unexpected results as indicated here.

Recommendation: Make sure `amount` fits in `uint128` using a `require` statement and use `hdiv` function with `uint128` parameter types.

QSP-9 Missing input validation
Severity: Informational
Status: Resolved
File(s) affected: HotColdWallet.sol

Description: The parameters of type `address` should be checked to be non-zero, before being assigned to state variables. The constructor of the HotColdWallet contract does not check if any of the 4 parameters of type `address` are different from 0x0, which could lead to transfers to 0x0.

Recommendation: Add checks for all parameters of type `address`.

QSP-10 Gas Usage / for Loop Concerns
Severity: Undetermined
Status: Resolved
File(s) affected: Loans.sol

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible. On L491-497, there is a for-loop iterating over collateralDepositFinalizedIndex. What are the expected max values for i? May the for loop consume too much gas?

Recommendation: Ensure that block gas-limits are considered for all functions that utilize loops.
QSP-11 Possible unhandled exception in `BitcoinCollateralProvider.setPaymentVariants()`

**Severity:** Undetermined

**Status:** Resolved

**File(s) affected:** `BitcoinCollateralProvider.js`, `BitcoinCollateralSwapProvider.js`

**Description:** The following line:
```javascript
if (col.colVout === undefined) { throw new Error('Could not find transaction based on redeem script') }
```

is present on L411 in `setPaymentVariants()` in `BitcoinCollateralSwapProvider.js`. However, it is missing in the same function in `BitcoinCollateralProvider.js` (L432-442).

**Exploit Scenario:**

**Recommendation:** Either ensure that this is intended, or add the conditional exception to `BitcoinCollateralProvider.setPaymentVariants()`.

Automated Analyses

**Mythril**

1. Myth warns that `ISPVRequestManager.fillRequest()` calls an external contract with a user-supplied address (corresponding to the consumer field of a request). However, all requests submitted in `Loans.sol` set this consumer field to the `Loans` contract itself, and therefore we consider this a false positive.

**Slither**

1. Slither reported several potential sources of re-entrancy, however the flagged external calls were to trusted contracts (e.g., Compound contracts), and as such we considered these false positives. Note that this relates to QSP-4 "Code does not adhere to the checks-effects-interactions pattern".

2. Slither detects that the `Oracle.med` state variable is not initialized, which may cause incorrect logic in `Oracle.setAssetPrice()` when `med.poke()` is invoked.

3. Slither detects that the `Oracle.reward()` function does not strictly adhere to the checks-effects-interactions pattern, since the `delete(asyncRequests[queryId])` occurs after the external token function calls. While this could theoretically lead to a reentrancy attack, it is unlikely to occur with token contracts that are trusted by all parties in the loan. Nonetheless, we recommend adjusting the code to adhere to the checks-effects-interaction pattern where possible.

4. Slither warns that the state variables `ChainLinkClient.link` is shadowed by `ChainLink.link`. We suggest renaming one of the variables to avoid shadowing.

**Adherence to Specification**

The code appears to meet the provided specification documents.

**Code Documentation**

The code in `atomicloans-eth-contracts` generally well-documented and provides doc-strings for most public functions. However, there are several functions and parameters that lack complete documentation which makes the intended semantics less clear:

1. The `rate` parameter of the `Funds.calcInterest()` function defined on L702 does not have a correct description in the code comments, which writes "@param rate The interest rate in seconds." It is not clear from this whether the value of `rate` is equal to the nominal annual interest rate in percentage terms or if it is `(1 + "the nominal annual interest rate in percentage terms")`. **Update:** fixed.

2. The second `Loans.cancel()` function on L617 is missing documentation and should be added, as the semantics appear different than the first `cancel()` function above. **Update:** fixed.

3. On L380 of `Loans.sol`, the comment mentions six uints, but the array has seven (`requestTimestamp` is missing in the comment). **Update:** fixed.

4. The comment on L23 in `Sales.sol` should say "Arbiter Signatures". **Update:** fixed.

5. The function `Loans.collateral()` defined on L221 is missing documentation. **Update:** fixed.

6. The function `Loans.minSeizableCollateralValue()` defined on L312 is missing documentation. **Update:** fixed.

7. The function `Loans.collateralValue()` on L318 is missing documentation. **Update:** fixed.

8. The function `Loans.minCollateralValue()` on L325 is missing documentation. **Update:** fixed.

9. The function `Loans.spv()` on L459 is missing documentation. **Update:** fixed.

10. The function `Loans.liquidate()` on L674 is missing documentation. **Update:** fixed.

11. The function `Sales.accept()` on L223 is missing documentation. **Update:** fixed.

12. The function `BitcoinCollateralProvider.getCollateralPaymentVariants()` on L176 is missing documentation. **Update:** fixed.

13. We further recommend adding documentation even for internal and private functions, such as `Loans.requestSpv()` and `Loans.close()`. **Update:** fixed.

14. The `create()` function defined on L383 of `Loans.sol` specifies the `bytes32 fundIndex` as its last parameter. This function is called on L730 of `Funds.sol`, where the last parameter given to this function call is `bytes32 fund`, which is specified to be "The Id of the Loan Fund", a slightly different term than `fundIndex`, which is declared as a global variable of the `Funds.sol` contract on L27. A clear distinction between "fund id" and
"fund index" should be made in the code comments of Funds.sol. Update: fixed.

15. The code comment on L163 of Loans.sol contains a "?" which makes the spec ambiguous. Update: fixed.

16. In general, there is less inline documentation in atomicloans-oracle-contracts. We recommend updating these contracts with improved docstrings. Update: fixed.

17. In HotColdWallet.sol, we recommend adding a docstring indicating that the bytes in isRequest() correspond to the signature of Funds.request(). Update: fixed.

**Adherence to Best Practices**

The code does not fully adhere to best practices. In particular:

1. On L50-87 of Loans.sol, the description of Loan does not fully match the struct. In particular, the requestTimestamp and closedTimestamp fields are not described, and the documented collateral field does not exist. Update: fixed.

2. In Loans.sol, all view functions from L177-294 do not first check if the Loan exists. Update: Since the functions will by default return zero if the loan does not exist, and adding a require-statement may interfere with dApp functionality, we consider this resolved.

3. Loans.constructor() does not check that the parameters are non-zero. Similarly, the setter functions setSales(), setP2WSH(), and setOnDemandSpv() should check that the parameters are non-zero. Update: fixed.

4. In Sales.sol, the constructor in L105 does not check if params are non-zero. Similarly, provideSig() does not check if refundableSig and seizableSig are non-zero. Update: fixed.

5. In Oracle.sol, the constructor does not check that the parameters are non-zero. Update: fixed.

6. The function Funds.update() does not check if arbiter_ is non-zero. A similar issue exists in create(), createCustom(), updateCustom(). A similar issue applies to the borrower_ parameter to the request() function on L540. Update: The Atomic Loans team has confirmed that the arbiter may be set to zero.

7. The function Loans.spv() does not check correctness of any inputs. It should be ensured that the invoking onDemandSpv contract is providing proper inputs. Update: fixed.

8. The function Loans.create() does not check correctness of inputs. Specifically:
   1. loanExpiration_ should be greater than now
   2. The borrower (usrs_[0]) and sender (usrs_[1]) should be checked to be non-zero
   3. The amounts in vals_ should be non-zero (except for the optional arbiter fee).
   4. It may also be appropriate to ensure that usrs_[0] != usrs_[1], and possibly that the penalty < collateral amount.
   5. Similar checks should be made in the functions createCustom() and updateCustom().
   6. Update: fixed.

9. In Funds.createCustom(), sanity constraints should be added to validate the input arguments, such as minLoanAmt_ <= maxLoanAmt_ and minLoanDur_ <= maxLoanDur_. Similar issues apply to updateCustom(). Update: fixed.

10. In Funds.sol, the constructor in L91 doesn’t check if params are non-zero. Similarly, the setter functions in L117-208 do not check if parameters are non-zero. Update: fixed.

11. In Funds.sol, the docstring for Fund on L51-67 contains extra fields custom and compoundEnabled, which should be associated with the bools struct instead. Update: fixed.

12. The function Loans.liquidate() does not check if secretHash and pubKeyHash are non-zero. Update: fixed.

13. There is a cyclic import between Medianizer.sol and Oracle.sol. We recommend refactoring the contracts to avoid the cyclic import. Update: fixed.

14. In Loans.minSeizableCollateralValue(), the variable set returned from med.peek() is not checked. We recommend adding require(set) before computing the collateral value. Update: fixed.

15. In atomicloans-oracle-contracts, functions that do not have an explicit access modifier (e.g., public, private, internal, etc.) should be explicitly labelled "public". Update: fixed.

16. On L63 of Medianizer.sol, uint256 should be used instead of uint. Update: fixed.

17. On L1923, the function Loans.setSecretHashes() is declared to return a bool, but has no return statement. Update: fixed.

18. There is missing input validation on Loans.setSecretHashes(), specifically the borrower and lender pubkeys. (Although, the lender/borrower should simply not fund/approve the loan if the details are incorrect.) Update: Atomic Loans has confirmed that they've implemented validation to ensure that the secret hashes provided are proper on the client side, so this should be fine.

19. On L69 of Loans.sol, there is no need for the if-statement. Update: fixed.

20. In Oracle.sol, there is an uninitialized constant Medianizer med. It is expected that contracts that inherit from Oracle should initialize the
constant in their constructor, but documentation should be added to ensure that this occurs. **Update**: fixed.

21. In **Funds.sol**, there is commented out code on L107-110 that should be removed. **Update**: fixed.

22. Lv7 of **Oracle.sol** is too long and difficult to parse. We recommend splitting it over multiple line of code. **Update**: fixed.

23. L73 in **Funds.sol** is too long (220 chars) and hard to follow because of the nested function calls. It would be easier to maintain if each of the 7 values in the array would be on a separate line. **Update**: fixed.

24. The function parameter names are inconsistent inside **Funds.sol**, because most parameter names end with an underscore "_" character, while a few parameter names do not end with this character, e.g., the fund parameter for all functions where it appears. Further, the amount parameter sometimes appears with an underscore (e.g., in the createLoan() function) and sometimes without an underscore at the end (e.g., in the calcInterest() function). The same applies to other parameters such as loanDur. This also happens in other files such as **Oracle.sol** and **Mediainizer.sol**. **Update**: fixed.

25. There are several TODOs present in code, which should be removed/resolved:

- L35 in **Loans.sol**
- L372 in **Loans.sol**. This line also contains commented out code.
- L41, L460-461, L492-293, L539 in **BitcoinCollateralProvider.js**
- L39, L430-431, L460-461 in **BitcoinCollateralSwapProvider.js**
- **Update**: fixed.

26. There are many `require` statements missing a string message containing the details/reason for the error. This would count as code comments and would help maintain the code. **Update**: fixed.

27. The `request()` function on L540 of **Funds.sol** does not check if the `secretHashes_` parameter actually contains 8 non-empty values. The caller could have forgot to set all 8 hash values by mistake. **Update**: fixed.

28. Tab and spaces are mixed in the code, e.g., L18-24 in **Sales.sol** use tabs at the beginning, while all other lines use spaces. **Update**: fixed.

29. The comment on L45 of **Sales.sol** is missing one or more words. **Update**: fixed.

30. `address` parameters in events should be indexed. For example, the following events have `address` parameters that are not indexed:

- L142 of **ChainlinkedTesting.sol**
- L145 of **ChainlinkedTesting.sol**
- L148 of **ChainlinkedTesting.sol**
- **Update**: The **ChainlinkedTesting.sol** contracts are simply a copied modified version of **Chainlinked.sol**, so these are less of a concern.

31. Code clones (copy-pasting of code) should be avoided:

- L195-199 from **BitcoinCollateralProvider.js** are duplicated on L220-224. This hints that the `getLockAddress()` function could be called inside the `lock` function to avoid this code duplication and ease maintenance.
- L232-235 from **BitcoinCollateralProvider.js** are duplicated on L243-246 and L254-257. These lines should be extracted into their own function.
- L369-392 from **BitcoinCollateralProvider.js** are duplicated on L400-421.
- **Update**: fixed.

32. Unused variables and constants should be removed:

- The value of `approveExpiration` assigned on L67, L446, L473 of **BitcoinCollateralProvider.js** is never used.
- The value of `network` assigned on L292, L321, L364, L398, L516 **BitcoinCollateralProvider.js** is never used.
- `arbiterPubKey` on L291, L320 in **BitcoinCollateralProvider.js** is never used.
- `arbiterPubKey` on L257, L288 in **BitcoinCollateralSwapProvider.js** is never used.
- **Update**: fixed.

33. On L25 and L28 of **BitcoinCollateralProvider.js**, the `changeHot` function does not check if the `newHot_` address is different from `hot`, i.e. the old hot wallet address. This means that calling `changeHot` with the same address of the current hot wallet would not fail, but merely waste gas. Is this intended? **Update**: fixed.

34. In **HotColdWallet.sol**, the `changeHot` function does not check if the `newHot_` address is different from `hot`, i.e. the old hot wallet address. This means that calling `changeHot` with the same address of the current hot wallet would not fail, but merely waste gas. Is this intended? **Update**: fixed.

35. In **HotColdWallet.sol**, the `isRequest()` function does not check if the length of the `data` bytes array is at least 4. If this check is performed inside `isRequest()` then the check on L28, namely if `(data.length > 0)` is no longer necessary. **Update**: fixed.

36. In **HotColdWallet.sol**, visibility of `funds`, `loans` and `sales` state variables should be made explicit.
Test Results

Test Suite Results

scriptNumSize
✓ should return 5 if value is greater than 0x7fffffff (45ms)
✓ should return 4 if value is greater than 0x7ffffff (38ms)
✓ should return 3 if value is greater than 0x7fff
✓ should return 2 if value is greater than 0x7f
✓ should return 1 if value is greater than 0x00
✓ should return 0 if value is 0x00

Contract: DAI Compound
deposit
✓ should update cBalance based on compound exchange rate of cTokens (4555ms)
✓ should update marketLiquidity to include interest gained from Compound (4913ms)
withdraw
✓ should update cBalance based on compound exchange rate of cTokens (5297ms)
✓ should update marketLiquidity to include interest gained from Compound (5452ms)
request
✓ should update cBalance based on compound exchange rate of cTokens (7960ms)
✓ should update marketLiquidity to include interest gained from Compound (7878ms)
enableCompound
✓ should properly convert DAI to cDAI at the current exchangeRate and update token and cToken balances (5421ms)
✓ should transfer tokenMarketLiquidity to cTokenMarketLiquidity at DAI to cDAI exchangeRate (5186ms)
disableCompound
✓ should properly convert cDAI to DAI at the current exchangeRate and update token and cToken balances (6243ms)
✓ should transfer cTokenMarketLiquidity to tokenMarketLiquidity at cDAI to DAI exchangeRate (6065ms)
setCompound
✓ should fail if called twice (190ms)

Contract: USDC Compound
deposit
✓ should update cBalance based on compound exchange rate of cTokens (6230ms)
✓ should update marketLiquidity to include interest gained from Compound (6818ms)
withdraw
✓ should update cBalance based on compound exchange rate of cTokens (4885ms)
✓ should update marketLiquidity to include interest gained from Compound (5698ms)
request
✓ should update cBalance based on compound exchange rate of cTokens (7400ms)
✓ should update marketLiquidity to include interest gained from Compound (8353ms)
enableCompound
✓ should properly convert DAI to cDAI at the current exchangeRate and update token and cToken balances (5239ms)
✓ should transfer tokenMarketLiquidity to cTokenMarketLiquidity at DAI to cDAI exchangeRate (6255ms)
disableCompound
✓ should properly convert cDAI to DAI at the current exchangeRate and update token and cToken balances (6620ms)
✓ should transfer cTokenMarketLiquidity to tokenMarketLiquidity at cDAI to DAI exchangeRate (6597ms)
setCompound
✓ should fail if called twice (176ms)

Contract: DAI End to end (BTC/ETH)
Regular loan flow with repayment before loanExpiration
✓ should request, lock, approve, withdraw, repay, accept, unlock (4964ms)
✓ should request, lock, approve, withdraw, liquidate, accept, claim (6871ms)
Liquidation on default
✓ should request, lock, approve, withdraw, wait until loanExpiration, liquidate, accept, claim (4470ms)
2 failed liquidations then claim
✓ should request, lock, approve, withdraw, wait until loanExpiration, liquidate, liquidate, liquidate, accept, claim (8250ms)
Seize after liquidation
✓ should request, lock, approve, withdraw, wait until loanExpiration, liquidate, liquidate, liquidate, liquidate, accept, claim (7837ms)

Contract: USDC End to end (BTC/ETH)
Regular loan flow with repayment before loanExpiration
✓ should request, lock, approve, withdraw, repay, accept, unlock (5472ms)
Liquidation when below 140% collateralization
✓ should request, lock, approve, withdraw, liquidate, accept, claim (5925ms)
Liquidation on default
✓ should request, lock, approve, withdraw, wait until loanExpiration, liquidate, accept, claim (5521ms)
2 failed liquidations then claim
✓ should request, lock, approve, withdraw, wait until loanExpiration, liquidate, liquidate, liquidate, liquidate, accept, claim (8264ms)
Seize after liquidation
✓ should request, lock, approve, withdraw, wait until loanExpiration, liquidate, liquidate, liquidate, accept, claim (8171ms)

Contract: DAI Funds
create
✓ should fail if user tries to create two loan funds (542ms)
✓ should succeed in updating non-custom loan fund (880ms)
✓ should fail in updating non-custom loan fund with > 10 years maxLoanDur (626ms)
✓ should fail in updating non-custom loan fund with > 10 years fundExpire (769ms)
✓ should fail creating loan fund with > 10 years fundExpire and maxLoanDur (187ms)
✓ should fail creating loan fund with 0 fundExpire and maxLoanDur (276ms)
✓ should succeed in withdrawing from non-custom loan fund (3819ms)
✓ should succeed with setCompoundEnabled false if compoundSet is false (1580ms)
✓ should succeed with setCompoundEnabled false if compoundSet is false (1396ms)
✓ should deposit funds on create if amount > 0 (1351ms)
createCustom
✓ should fail if user tries to create two loan funds (254ms)
✓ should fail creating custom loan fund with 18 years+ fundExpire and maxLoanDur (105ms)
✓ should fail creating custom loan fund with 0 fundExpire and maxLoanDur (102ms)
✓ should succeed with setCompoundEnabled false if compoundSet is false (1464ms)
✓ should fail creating loan fund with 0 fundExpire and maxLoanDur (1114ms)
**Contract: USDC Funds**

**create**
- should fail if user tries to create two loan funds (202ms)
- should succeed in creating a loan fund with maxLoanDur > 10 years (288ms)
- should fail creating loan fund with > 10 years fundExpiry (271ms)
- should succeed in creating loan fund with > 10 years fundExpiry and maxLoanDur (92ms)
- should fail if user tries to create two loan funds (266ms)
- should succeed in creating loan fund with > 10 years fundExpiry and maxLoanDur (229ms)
- should fail creating loan fund with 0 fundExpiry and maxLoanDur (129ms)
- should succeed in creating loan fund with > 10 years fundExpiry and maxLoanDur (142ms)
- should fail if user tries to create two loan funds (1045ms)

**createCustom**
- should fail if user tries to create two loan funds (202ms)
- should fail creating custom loan fund with 10 years+ fundExpiry and maxLoanDur (229ms)
- should fail creating loan fund with 0 fundExpiry and maxLoanDur (129ms)
- should succeed in creating loan fund with > 10 years fundExpiry and maxLoanDur (142ms)
- should fail if user tries to create two loan funds (1045ms)

**deposit**
- should fail depositing to fund if erc20 allowance less than amount (1337ms)
- should update cTokenMarketLiquidity if not custom and compoundEnabled (1608ms)
- should not update cTokenMarketLiquidity if custom and compoundEnabled (947ms)

**generate secret hashes**
- should push secrets hashes to secretHashes for user address (313ms)
- should fail trying to return incorrect secretHashes index (1337ms)

**push funds**
- should allow anyone to push funds to loan fund (383ms)
- should request and complete loan successfully if loan setup correctly (2524ms)
- should increment fundIndex (292ms)

**set fund details**
- should allow changing of pubk (142ms)
- should allow changing of fund details (363ms)
- should fail changing of fund details with > 10 years+ fundExpiry and maxLoanDur (119ms)

**update**
- should fail if not called by lender (278ms)
- should fail if not called by lender (126ms)
- should fail if trying to update non-custom fund (287ms)

**request**
- should fail if msg.sender is not lender (323ms)
- should fail if balance is less than amount requested (487ms)
- should fail if amount is less than min loan amount (337ms)
- should fail if amount is greater than max loan amount (358ms)
- should succeed requesting from non-custom fund (4926ms)

**withdraw funds**
- should withdraw funds successfully if called by owner (837ms)
- should allow withdrawing to a specific address as long as it's called by the owner of the fund (1083ms)

**maxFundDuration**
- should succeed if expiry of Fund is set after loan request (2458ms)
- should fail if expiry of Fund is set before loan request (891ms)

**ensureNotZero**
- should convert 0 to MAX_LOAN_LENGTH if addNow bool is false (41ms)
- should convert 0 to MAX_LOAN_LENGTH + now if addNow bool is true (150ms)

**setLoans**
- should not allow setLoans to be called twice (94ms)
- should fail if setLoans to be called by address other than deployer (90ms)
- should fail if token is pausable and paused (1073ms)

**setCompound**
- should not allow setCompound to be called by address other than deployer (88ms)

**setUtilizationInterestDivisor**
- should fail if set by non deployer (120ms)
- should set utilizationInterestDivisor if called by deployer (98ms)

**setMaxUtilizationDelta**
- should fail if set by non deployer (86ms)
- should set maxUtilizationDelta if called by deployer (108ms)

**setGlobalInterestRate**
- should fail if set by non deployer (77ms)
- should set globalInterestRate if called by deployer (100ms)

**setMaxInterestRateNumerator**
- should fail if set by non deployer (76ms)
- should set maxInterestRateNumerator if called by deployer (110ms)

**setMinInterestRateNumerator**
- should fail if set by non deployer (90ms)
- should set minInterestRateNumerator if called by deployer (98ms)

**setInterestUpdateDelay**
- should fail if set by non deployer (92ms)
- should set interestUpdateDelay if called by deployer (112ms)

**setDefaultArbiterFee**
- should fail if set by non deployer (80ms)
- should set defaultArbiterFee if called by deployer (97ms)
- should fail trying to set default arbiter fee > 1% (84ms)

**secretHashesCount**
- should return the secret hash count for a specific address (150ms)

**enableCompound**
- should fail if compoundSet is false (1109ms)
- should fail if compound already enabled (397ms)
- should fail if msg.sender isn't lender (290ms)

**disableCompound**
- should fail if compound isn't already enabled (288ms)
- should fail if msg.sender isn't lender (320ms)

**decreaseTotalBorrow**
- should fail calling if not loans contract address (95ms)
- should fail decreaseTotalBorrow if totalBorrow is 0 (1566ms)

**Contract: USDC Funds**

**create**
- should fail if user tries to create two loan funds (202ms)
- should succeed in creating a loan fund with > 10 years maxLoanDur (288ms)
- should fail in creating loan fund with > 10 years fundExpiry (271ms)
- should succeed in creating loan fund with > 10 years fundExpiry and maxLoanDur (92ms)
- should fail if user tries to create two loan funds (266ms)
- should succeed in creating loan fund with > 10 years fundExpiry and maxLoanDur (229ms)
- should fail creating loan fund with 0 fundExpiry and maxLoanDur (129ms)
- should succeed in creating loan fund with > 10 years fundExpiry and maxLoanDur (142ms)
- should fail if user tries to create two loan funds (1045ms)

**createCustom**
- should fail if user tries to create two loan funds (202ms)
- should fail creating custom loan fund with > 10 years fundExpiry and maxLoanDur (229ms)
- should fail creating custom loan fund with 0 fundExpiry and maxLoanDur (129ms)
- should succeed in creating loan fund with > 10 years fundExpiry and maxLoanDur (142ms)
- should fail if user tries to create two loan funds (1045ms)
✓ should fail depositing to fund if erc20 allowance less than amount (2394ms)
✓ should update cTokenMarketLiquidity if not custom and compoundEnabled (1529ms)
✓ should not update cTokenMarketLiquidity if custom and compoundEnabled (1200ms)

generate secret hashes
✓ should push secrets hashes to secretHashes for user address (447ms)
✓ should fail trying to return incorrect secretHashes index

push funds
✓ should allow anyone to push funds to loan fund (538ms)
✓ should request and complete loan successfully if loan setup correctly (3106ms)

opening loan fund
✓ should increment fundIndex (457ms)

set fund details
✓ should allow changing of pubk (138ms)
✓ should allow changing of fund details (324ms)
✓ should fail changing of fund details with 10 years+ fundExpiry and maxLoanDur (125ms)

update
✓ should fail if not called by lender (284ms)
✓ should fail if not called by lender (176ms)
✓ should fail if trying to update non-custom fund (369ms)

request
✓ should fail if msg.sender is not lender (383ms)
✓ should fail if balance is less than amount requested (470ms)
✓ should fail if amount is less than min loan amount (455ms)
✓ should fail if amount is greater than max loan amount (904ms)
✓ should succeed requesting from non-custom fund (8042ms)

withdraw funds
✓ should withdraw funds successfully if called by owner (837ms)
✓ should fail withdrawing funds if not called by owner (688ms)
✓ should allow withdrawing to a specific address as long as it's called by the owner of the fund (817ms)

maxFundDuration
✓ should succeed if expiry of Fund is set after loan request (1831ms)
✓ should fail if expiry of Fund is set before loan request (984ms)

ensureNotZero
✓ should convert 0 to MAX_LOAN_LENGTH if addNow bool is false (42ms)
✓ should convert 0 to MAX_LOAN_LENGTH + now if addNow bool is true (54ms)

setLoans
✓ should not allow setLoans to be called twice (187ms)
✓ should not allow setLoans to be called by address other than deployer (83ms)
✓ should fail if token is paused and paused (591ms)

setCompound
✓ should not allow setLoans to be called by address other than deployer (103ms)

setUtilizationInterestDivisor
✓ should fail if set by non deployer (100ms)
✓ should set utilizationInterestDivisor if called by deployer (119ms)

setMaxUtilizationDelta
✓ should fail if set by non deployer (102ms)
✓ should set maxUtilizationDelta if called by deployer (204ms)

setGlobalInterestRateNumerator
✓ should fail if set by non deployer (85ms)
✓ should set maxUtilizationDelta if called by deployer (135ms)

setGlobalInterestRate
✓ should fail if set by non deployer (95ms)
✓ should set globalInterestRate if called by deployer (119ms)

setMaxInterestRateNumerator
✓ should fail if set by non deployer (90ms)
✓ should set globalInterestRate if called by deployer (116ms)

setMinInterestRateNumerator
✓ should fail if set by non deployer (146ms)
✓ should set globalInterestRate if called by deployer (141ms)

setInterestUpdateDelay
✓ should fail if set by non deployer (133ms)
✓ should set globalInterestRate if called by deployer (100ms)

setDefaultArbiterFee
✓ should fail if set by non deployer (185ms)
✓ should set defaultArbiterFee if called by deployer (120ms)
✓ should fail trying to set default arbiter fee > 1% (81ms)

secretHashesCount
✓ should return the secret hash count for a specific address (178ms)

enableCompound
✓ should fail if compoundSet is false (2074ms)
✓ should fail if compound already enabled (362ms)
✓ should fail if msg.sender isn't lender (424ms)

disableCompound
✓ should fail if compound isn't already enabled (301ms)
✓ should fail if msg.sender isn't lender (302ms)

increaseTotalBorrow
✓ should fail calling if not loans contract address (110ms)
✓ should fail increaseTotalBorrow if totalBorrow is 0 (138ms)

Contract: ALCompound
getComptrollerAddress
✓ should return current comptroller address

Contract: HotColdWallet
Constructor
✓ should allow creation of fund (313ms)

callFunds
✓ should succeed if cold wallet (761ms)
✓ should succeed if hot wallet and requesting loan (1770ms)
✓ should fail if hot wallet (141ms)

callLoans
✓ should succeed if called by hot wallet (1891ms)
✓ should fail if not hot or cold wallet (2285ms)

changeSales
✓ should succeed if called by hot wallet (4797ms)
✓ should fail if not hot or cold wallet (4697ms)

ChangeHot
✓ should succeed in changing hot wallet if from cold wallet (121ms)
✓ should fail if not cold wallet (85ms)
✓ should fail if new hot address is null (87ms)

Contract: DAI Global Interest Rate Decrease
  global interest rate
  ✓ should increase global interest rate after a day if utilization ratio increases (12965ms)

Contract: USDC Global Interest Rate Decrease
  global interest rate
  ✓ should increase global interest rate after a day if utilization ratio increases (12268ms)

Contract: DAI Global Interest Rate Increase
  global interest rate
  ✓ should increase global interest rate after a day if utilization ratio increases (27527ms)

Contract: USDC Global Interest Rate Increase
  global interest rate
  ✓ should increase global interest rate after a day if utilization ratio increases (30271ms)

Contract: DAI Loans
  constructor
  ✓ should fail deploying Loans if token is pausable and paused (1175ms)

setSales
  ✓ should fail if msg.sender is not deployer (957ms)
  ✓ should fail if token address already set (1648ms)

setOnDemandSpv
  ✓ should fail if msg.sender is not deployer (1191ms)
  ✓ should fail if onDemandSpv already set (1244ms)
  ✓ should fail if onDemandSpv is null (1888ms)

unsetOnDemandSpv
  ✓ should fail if msg.sender is not deployer (1921ms)
  ✓ should fail if onDemandSpv has not been set already (1219ms)
  ✓ should allow loan to be created and repaid without onDemandSpv set (4952ms)

setCollateral
  ✓ should fail if msg.sender is not deployer (1143ms)
  ✓ should fail if Collateral has already been set (1598ms)
  ✓ should fail if Collateral address to be set is null (1492ms)

Collateral.setCollateral
  ✓ should fail if not called by loans contract (1225ms)

create
  ✓ should fail if fund lender address does not match provided lender address (183ms)
  ✓ should fail if requestTimestamp is duplicated (251ms)

setSecretHashes
  ✓ should fail calling twice (750ms)
  ✓ should fail if called by address which is not the borrower, lender, or funds contract address (1026ms)

fund
  ✓ should fail if secret hashes not set (624ms)
  ✓ should fail if called twice (1031ms)
  ✓ should fail if using pausable token that is paused (2706ms)

approve
  ✓ should fail if not funded (889ms)
  ✓ should fail if msg.sender is not lender (952ms)
  ✓ should fail if after current time is after approveExpiration (1823ms)

repay
  ✓ should fail if loan is already off (2753ms)
  ✓ should fail if liquidation has started (3274ms)
  ✓ should fail if not withdrawn (831ms)
  ✓ should fail if after loanExpiration (3478ms)
  ✓ should fail if amount is more than owedForLoan (1810ms)

accept
  ✓ should accept successfully if lender secret provided (2186ms)
  ✓ should accept successfully if arbiter secret provided (2720ms)
  ✓ should accept successfully and send funds directly to lender if fundId is 0 (4352ms)
  ✓ should fail if loan is already accepted (2778ms)
  ✓ should fail if withdrawn and not repaid (1432ms)
  ✓ should fail if msg.sender is not lender or arbiter (2441ms)
  ✓ should fail if secret does not hash to secretHashB1 or secretHashC1 (2108ms)
  ✓ should fail if current time is greater than acceptExpiration (2426ms)

cancel
  ✓ should successfully cancel loan and return funds to loan fund (337ms)
  ✓ should successfully cancel loan without secret if after seizureExpiration (598ms)
  ✓ should fail if loan is already accepted (2140ms)
  ✓ should fail if not withdrawn (195ms)
  ✓ should fail if current time is less than seizureExpiration and no secret is provided (214ms)
  ✓ should fail if already liquidated (315ms)

refund
  ✓ should return loan repayment to borrower (2918ms)
  ✓ should return loan repayment to borrower with non-custom fund (5575ms)
  ✓ should fail if loan is already accepted (2523ms)
  ✓ should fail if loan has been liquidated (3896ms)
  ✓ should fail if before acceptExpiration (2246ms)
  ✓ should fail if not repaid (2412ms)
  ✓ should fail if msg.sender != borrower (2974ms)

getters
  ✓ should add borrower to borrowerLoans list after requesting loan (42ms)
  ✓ should add lender to lenderLoans list after requesting loan (48ms)

liquidate
  ✓ should be safe if above liquidation ratio (131ms)
  ✓ should succeed at creating a sale if below liquidation ratio (4131ms)

default
  ✓ should fail if current time before loan expiration (1769ms)
  ✓ should allow for liquidation to start if loan is defaulted (2644ms)

withdraw
  ✓ should fail trying to withdraw twice (1729ms)
  ✓ should fail if loan is off (2585ms)
  ✓ should fail if not funded (719ms)
  ✓ should fail if not approved (74ms)
  ✓ should fail if secret provided does not hash to secretHashA1 (280ms)
✓ should fail if token is pausable and paused (2788ms)
setSales
✓ should not allow setSales to be called twice (100ms)
borrower
✓ should return borrower address
lender
✓ should return lender address
arbiter
✓ should return arbiter address
owing
✓ should return principal + interest + fee when first requested (90ms)
✓ should return principal + interest + fee - repaid if parts of the loan were repaid (2243ms)
funded
✓ should return boolean determining whether funds have been deposited into loan (879ms)
✓ should return boolean determining whether funds have been deposited into loan (820ms)
approved
✓ should return boolean determining whether loan has been approved (1114ms)
withdrawn
✓ should return boolean determining whether loan has been withdrawn (1314ms)
paid
✓ should return boolean determining whether loan has been repaid (1771ms)
minCollateralValue
✓ should return 0 if repaid (1703ms)
minSeizableCollateral
✓ should change upon partial repayment (2041ms)
Contract: USDC Loans
constructor
✓ should fail deploying Loans if token is pausable and paused (676ms)
setSales
✓ should fail if msg.sender is not deployer (1195ms)
setP2WSH
✓ should fail if msg.sender is not deployer (1294ms)
✓ should fail if p2wsh already set (1142ms)
setOnDemandSpv
✓ should fail if msg.sender is not deployer (1804ms)
✓ should fail if onDemandSpv already set (1614ms)
✓ should fail if onDemandSpv is null (1930ms)
unsetOnDemandSpv
✓ should fail if msg.sender not deployer (1208ms)
✓ should fail if onDemandSpv has not been set already (1350ms)
✓ should set onDemandSpv address to null (1755ms)
setCollateral
✓ should fail if msg.sender is not deployer (1280ms)
✓ should fail if Collateral has already been set (1804ms)
✓ should fail if Collateral address to be set is null (1012ms)
Collateral.setCollateral
✓ should fail if not called by loans contract (1271ms)
create
✓ should fail if fund lender address does not match provided lender address (216ms)
✓ should fail if requestTimestamp is duplicated (180ms)
setSecretHashes
✓ should fail calling twice (691ms)
✓ should fail if called by address which is not the borrower, lender, or funds contract address (528ms)
fund
✓ should fail if secret hashes not set (615ms)
✓ should fail if called twice (911ms)
✓ should fail if using pausable token that is paused (2113ms)
approve
✓ should fail if not funded (778ms)
✓ should fail if msg.sender is not lender (1371ms)
✓ should fail if after current time is after approveExpiration (1280ms)
repay
✓ should fail if loan is already off (2611ms)
✓ should fail if liquidation has started (2668ms)
✓ should fail if not withdrawn (273ms)
✓ should fail if after loanExpiration (4817ms)
✓ should fail if amount is more than owedForLoan (2356ms)
accept
✓ should accept successfully if lender secret provided (3128ms)
✓ should accept successfully if arbiter secret provided (2780ms)
✓ should accept successfully and send funds directly to lender if fundId is 0 (3237ms)
✓ should fail if loan is already accepted (2685ms)
✓ should fail if withdrawn and not repaid (1362ms)
✓ should fail if msg.sender is not lender or arbiter (2562ms)
✓ should fail if secret does not hash to secretHashB1 or secretHashC1 (3042ms)
✓ should fail if current time is greater than acceptExpiration (2295ms)
cancel
✓ should successfully cancel loan and return funds to loan fund (4805ms)
✓ should successfully cancel loan without secret if after seizureExpiration (651ms)
✓ should fail if loan is already accepted (2857ms)
✓ should fail if not withdrawn (262ms)
✓ should fail if current time is less than seizureExpiration and no secret is provided (221ms)
✓ should fail if already liquidated (3435ms)
refund
✓ should return loan repayment to borrower (2181ms)
✓ should return loan repayment to borrower with non-custom fund (5638ms)
✓ should fail if loan is already accepted (2880ms)
✓ should fail if loan has been liquidated (4087ms)
✓ should fail if before acceptExpiration (2254ms)
✓ should fail if not repaid (2079ms)
✓ should fail if msg.sender != borrower (2015ms)
getters
✓ should add borrower to borrowerLoans list after requesting loan (46ms)
✓ should add lender to lenderLoans list after requesting loan (42ms)
liquidate
✓ should be safe if above liquidation ratio (1680ms)
✓ should succeed at creating a sale if below liquidation ratio (4292ms)
default
✓ should fail liquidation if current time before loan expiration (1735ms)
✓ should allow for liquidation to start if loan is defaulted (1263ms)

withdraw
✓ should fail trying to withdraw twice (1168ms)
✓ should fail if loan is off (1753ms)
✓ should fail if not funded (721ms)
✓ should fail if not approved (102ms)
✓ should fail if secret provided does not hash to secretHashA1 (234ms)
✓ should fail if token is paused and paused (2873ms)

setSales
✓ should not allow setSales to be called twice (100ms)

borrower
✓ should return borrower address
✓ should return lender address

arbiter
✓ should return arbiter address

owing
✓ should return principal + interest + fee when first requested (114ms)
✓ should return principal + interest + fee - repaid if parts of the loan were repaid (1723ms)

funded
✓ should return boolean determining whether funds have been deposited into loan (1412ms)
✓ should return boolean determining whether funds have been deposited into loan (926ms)

approved
✓ should return boolean determining whether loan has been approved (150ms)

withdrawn
✓ should return boolean determining whether loan has been withdrawn (1139ms)

paid
✓ should return boolean determining whether loan has been repaid (2131ms)

minCollateralValue
✓ should return 0 if repaid (2080ms)
minSeizableCollateral
✓ should return 0 if repaid (2080ms)

Contract: DAI P2WSH
Should generate proper p2wsh
✓ should generate p2wsh for refundable and seizable collateral (1771ms)

Contract: DAI Sales
3 liquidations
✓ should allow for 3 liquidations before considered failed (4060ms)
✓ should fail if liquidation called before previous liquidation is finished (1800ms)

create
✓ should fail if msg.sender isn't loans contract address (86ms)

accept
✓ should disperse funds to rightful parties after partial repayment (3005ms)
✓ should disperse all funds to lender and arbiter if discountBuy + repaid doesn't cover principal + interest (3074ms)
✓ should disperse all funds to lender and arbiter if discountBuy + repaid covers only principal + interest + fee (3288ms)
✓ should disperse all remaining funds to medianizer if funds have been paid to lender and arbiter but not enough is needed to pay the penalty for the medianizer (3486ms)
✓ should disperse funds to lender, arbiter, and medianizer if there is enough funds for owedToLender, fee and penalty but not enough for borrower (2650ms)
✓ should disperse funds to rightful parties after partial repayment using provideSecretsAndAccept function (3411ms)
✓ should fail if accepted (2941ms)
✓ should fail if off (2344ms)
✓ should fail if hasSecret is false (4830ms)
✓ should fail if secret D is not revealed (5169ms)

provideSig
✓ should allow parties to sign and retrieve their signatures (2452ms)
✓ should fail if providing signature for incorrect sale (95ms)
✓ should fail if msg.sender isn't borrower, lender, or arbiter (1256ms)
✓ should fail if current time is greater than settlementExpiration (150ms)
✓ should fail if refundableSig is null (1649ms)
✓ should fail if seizableSig is null (1685ms)

hasSecrets
✓ should return 0 if no secrets provided (3004ms)

refund
✓ should refund if not off, not accepted, current time greater than settlementExpiration and discountBuy set (2518ms)
✓ should refund borrower repaid amount after 3rd liquidation attempt (6482ms)
✓ should fail refunding if already refunded (3119ms)
✓ should fail refunding if current time before settlement expiration (2599ms)
✓ should fail refunding if discountBuy already accepted (3425ms)
✓ should fail if discountBuy is 0 (181ms)

provideSecret
✓ should fail if sale not set (74ms)

Contract: USDC Sales
3 liquidations
✓ should allow for 3 liquidations before considered failed (4214ms)
✓ should fail if liquidation called before previous liquidation is finished (1855ms)

create
✓ should fail if msg.sender isn't loans contract address (90ms)

accept
✓ should disperse funds to rightful parties after partial repayment (3403ms)
✓ should disperse all funds to lender and arbiter if discountBuy + repaid doesn't cover principal + interest (3385ms)
✓ should disperse all funds to lender and arbiter if discountBuy + repaid covers only principal + interest + fee (3501ms)
✓ should disperse all remaining funds to medianizer if funds have been paid to lender and arbiter but not enough is needed to pay the penalty for the medianizer (3998ms)
✓ should disperse funds to lender, arbiter, and medianizer if there is enough funds for owedToLender, fee and penalty but not enough for borrower (2757ms)
✓ should disperse funds to rightful parties after partial repayment using provideSecretsAndAccept function (1872ms)
✓ should fail if accepted (2947ms)
✓ should fail if off (3102ms)
✓ should fail if hasSecrets is false (5159ms)
✓ should fail if secret D is not revealed (5073ms)
provideSig
✓ should allow parties to sign and retrieve their signatures (2757ms)
✓ should fail providing signature for incorrect sale (92ms)
✓ should fail if msg.sender isn't borrower, lender, or arbiter (1598ms)
✓ should fail if current time is greater than settlementExpiration (1610ms)
✓ should fail if refundableSig is null (1578ms)
✓ should fail if seizableSig is null (1228ms)
hasSecrets
✓ should return 0 if no secrets provided (4908ms)
refund
✓ should refund if not off, not accepted, current time greater than settlementExpiration and discountBuy set (2615ms)
✓ should refund borrower repaid amount after 3rd liquidation attempt (5931ms)
✓ should fail refunding if already refunded (2779ms)
✓ should fail refunding if current time before settlement expiration (2749ms)
✓ should fail refunding if discountBuy already accepted (2923ms)
✓ should fail if discountBuy is 0 (583ms)
provideSecret
✓ should fail if sale not set (93ms)

Contract: DAI Spv
Non-zero spv values
✓ should fail if zero values are provided to spv (94ms)
Add seizable collateral
✓ should update collateral value after 1 confirmation (8994ms)
Add refundable collateral
✓ should update collateral value after 1 confirmation if min seizable collateral value is satisfied (7711ms)
✓ should not update collateral value after 1 confirmation if min seizable collateral value is not satisfied (7854ms)
Add refundable collateral first then seizable collateral
✓ should not update collateral value until seizable collateral has been confirmed when min seizable collateral isn't satisfied after 1 confirmation (9854ms)
✓ should not update collateral value until seizable collateral has been confirmed when min seizable collateral isn't satisfied after 6 confirmations (9805ms)
Add seizable collateral first then refundable collateral
✓ should update collateral value as soon as seizable collateral is confirmed, and then increase it again once refundable collateral is added (8359ms)
Add seizable and refundable collateral multiple times with request confirmations out of order
✓ should not add collateral value if minSeizableCollateral is not satisfied (14482ms)
✓ should fail adding refundable collateral that is slightly above 1%, after seizable collateral has been added with 6 confirmations (9522ms)
✓ should fail adding refundable collateral that is slightly above 1%, after seizable collateral has been added with only 1 confirmation (9280ms)
Incorrect vout from onDemandSpv service
✓ should fail adding collateral if vout does not correspond to correct p2wsh (4912ms)
Incorrect onDemandSpv service address
✓ should fail adding collateral if onDemandSpv service address is incorrect (4144ms)
Collateral Balance
✓ should return the refundable + seizable collateral when no collateral has been added (2459ms)
✓ should return the refundable + seizable when minSeizableCollateral is not satisfied (4348ms)
✓ should return the refundable + seizable when temporaryCollateral expiration is past the 4 hour expiry date (4390ms)
✓ should return the refundable + seizable + temporary refundable + temporary seizable when adding collateral in queue while satisfying both minSeizableCollateral and temporaryCollateral expiration (5980ms)
Liquidation
✓ should succeed if below minimum collateralization ratio and refundable collateral added does not satisfy the minSeizableCollateral (6107ms)
✓ should fail if now is less than added temporary collateral expiration, minimum collateralization ratio is satisfied as well as minSeizableCollateral (5402ms)

425 passing (40m)

Contract: Chainlink
pack
✓ should fail if trying to pack twice before 15 minutes is up (570ms)
✓ should succeed in updating price of called once (511ms)
✓ should reward correct based on max (612ms)

Contract: Medianizer
fund
✓ should send funds to all oracle contracts
✓ should not return median for oracles if less than 5 are set (1374ms)
✓ should return correct median of oracles when only 5 are set (2166ms)
✓ should return correct median of all oracles when all oracles have same price (3939ms)
✓ should return correct median of all oracles when different prices (4589ms)
✓ should return median for oracles if 12 hours has passed since last update (4374ms)
✓ should fail if amount is greater than 2**128-1 (46ms)
compute
✓ should return median price of 0 if all oracle values are equal or above 2**128 (2445ms)
✓ should return median price of 2**128-1 if all oracle values are equal to 2**128-1 (2412ms)

Contract: Oraclize
pack
✓ should fail if trying to pack twice before 15 minutes is up (322ms)
✓ should succeed in updating price of called once (374ms)
✓ should reward correctly (325ms)
✓ should not reward if price has not changed by 1% (764ms)
setGasLimit
✓ should properly set gas limit (245ms)

17 passing (39s)

Client methods without providers
constructor
✓ should throw error if constructed with incorrect script
✓ should throw error if constructed with incorrect address
setPaymentVariants
✓ should throw if colVout undefined
buildFullColTx
✓ should create tx with default fee value if estimateFees is false

Client methods without providers
getCollateralOutput
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
✓ should throw NoProviderError
21 passing (31ms)

Code Coverage

<table>
<thead>
<tr>
<th>File</th>
<th>% Stmts</th>
<th>% Branch</th>
<th>% Funcs</th>
<th>% Lines</th>
<th>Uncovered Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>contracts/</td>
<td>98.78</td>
<td>82.77</td>
<td>99.32</td>
<td>99.13</td>
<td></td>
</tr>
<tr>
<td>Bytes.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Collateral.sol</td>
<td>99.05</td>
<td>82.26</td>
<td>100</td>
<td>99.06</td>
<td>280</td>
</tr>
<tr>
<td>CollateralInterface.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Funds.sol</td>
<td>99.61</td>
<td>84.24</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>FundsInterface.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>HotColdWallet.sol</td>
<td>100</td>
<td>83.33</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Loans.sol</td>
<td>98.01</td>
<td>81.55</td>
<td>98.08</td>
<td>97.98</td>
<td>274,788,791,802</td>
</tr>
<tr>
<td>LoansInterface.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Medianizer.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>P2WSH.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>P2WSHInterface.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Sales.sol</td>
<td>97.2</td>
<td>79.76</td>
<td>100</td>
<td>98.97</td>
<td>311</td>
</tr>
<tr>
<td>SalesInterface.sol</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>All files</td>
<td>98.78</td>
<td>82.77</td>
<td>99.32</td>
<td>99.13</td>
<td></td>
</tr>
</tbody>
</table>

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

d72b5cf95b4c2d801c2ed573e26f2c22004dc3bb3dabf9624a302384e6a39755 ./HotColdWallet.sol
f601587ed7d6ae4a3ffbf2765983ec8637b43218b78a8ec193643759142780d87f ./FundsInterface.sol
a6313aae62893db39ae647614dd58bb7a1499b8262212688f8a3d48915b96368 ./WETH9.sol
9973b9a476caea4b2f75ad738637ccal1ff420c28d1b2dc53831e1abc3df16d ./ERC20.sol
About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp’s team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $1B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp’s collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.

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.

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, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ 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 the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim 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. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, referred by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. 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. 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.