Quantstamp Security Assessment Certificate

July 2nd 2020 — Quantstamp Verified

Atomic Loans

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

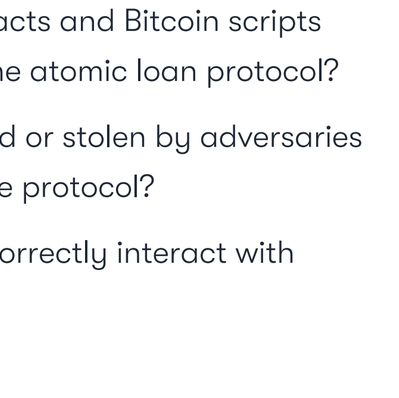
Executive Summary

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
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Atomic Loans Wiki Atomic Loans: Cryptocurrency Debt Instruments
Source Code	

Repository	Commit
atomicloans-eth-contracts	10c2493
atomicloans-oracle-contracts	06c90e0
chainabstractionlayer-loans	98ea474

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, 3f963fe, 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 `atomiclaons-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 `chainabstractionlayer-loans`, and 67d3df1 of `atomicloans-oracle-contracts`. 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 added a `HotColdWallet` smart contract in commit 8016c19. Only one informational issue was found.
Update 3: The Atomic Loans team has addressed our comments as of commit 878917d.

ID	Description	Severity	Status
QSP-1	Testing code included amongst main code repository	Low	Resolved
QSP-2	Loans can be requested for extremely long periods of time	Low	Resolved
QSP-3	Code does not adhere to the checks-effects-interactions pattern	Informational	Resolved
QSP-4	Unlocked Pragma	Informational	Resolved
QSP-5	Centralization of Power	Informational	Resolved
QSP-6	Undocumented magic constants	Informational	Resolved
QSP-7	Integer Overflow / Underflow	Informational	Resolved
QSP-8	Downcasting and Upcasting may lead to unexpected results	Informational	Resolved
QSP-9	Missing input validation	Informational	Resolved
QSP-10	Gas Usage / `for` Loop Concerns	Undetermined	Resolved
QSP-11	Possible unhandled exception in `BitcoinCollateralProvider.setPaymentVariants()`	Undetermined	Resolved

Quantstamp Audit Breakdown

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 L365/374 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 L689, there is a constant `3`.
3. In `Sales.sol` on L278, there is a constant `3`.
4. In `Funds.sol` on L121, 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 `1000000000937303470807876289` represents 3%.
6. In `Funds.sol` on L104, it is not clear if `1000000000236936036262880196` represents 0.75%.
7. In `Funds.sol` on L206, it is not clear if `1000000000315522921573372069` 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 `1000000000000000000` is used. Similarly, L21 and L31 use the value `2`.
10. In `CoinMarketCap.sol`, on L25 and L40 the value `1000000000000000000` 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 `1000000000000000000` is used. Similarly, L21 and L31 use the value `2`.
12. In `Gemini.sol`, L20 and L30 use the value `1000000000000000000`, and L21 and L31 use the value `2`.
13. In `SoChain.sol`, L20 and L30 use the value `1000000000000000000`, and L21 and L31 use the value `2`.
14. In `BitcoinCollateralProvider.js` L462 and L499 use the values `6` and `14`.

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`.

```
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 `if (col.colVout === undefined) { throw new Error('Could not find transaction based on redeem script') }` is present on L411 in the `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 L60-87 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 `Oraclize.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 L423, 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 L654 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. L47 of `Oracle.sol` is too long and difficult to parse. We recommend splitting it over multiple line of code. **Update:** fixed.

23. L733 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 `Medianizer.sol`. **Update:** fixed.

25. There are several TODOs present in code, which should be removed/resolved:
    1. L35 in `Loans.sol`.
    2. L372 in `Loans.sol`. This line also contains commented out code.
    3. L41, L460-461, L492-293, L539 in `BitcoinCollateralProvider.js`.
    4. L39, L430-431, L460-461 in `BitcoinCollateralSwapProvider.js`.
    5. **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:
    1. `NewOwner` on L142 of `ChainlinkedTesting.sol`
    2. `Transfer` on L145 of `ChainlinkedTesting.sol`
    3. `NewResolver` on L148 of `ChainlinkedTesting.sol`
    4. **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:
    1. 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.
    2. L232-235 from `BitcoinCollateralProvider.js` are duplicated on L243-246 and L254-257. These lines should be extracted into their own function.
    3. L369-390 from `BitcoinCollateralProvider.js` are duplicated on L400-421.
    4. **Update:** fixed.

32. Unused variables and constants should be removed:
    1. The value of `approveExpiration` assigned on L67, L446, L473 of `BitcoinCollateralProvider.js` is never used.
    2. The value of `network` assigned on L292, L321, L364, L398, L516 `BitcoinCollateralProvider.js` is never used.
    3. `arbiterPubKey` on L291, L320 in `BitcoinCollateralProvider.js` is never used.
    4. `arbiterPubKey` on L257, L288 in `BitcoinCollateralSwapProvider.js` is never used.
    5. **Update:** fixed.

33. On L25 and L28 of `BitcoinCollateralProvider.js`, the error messages are ambiguous because both of them mention that the “Mode must be one of”, but the values that follow are different. If a user sees the first message, then fixes the problem and then sees the second message, it would create confusion. It is recommended to add the word “script” after “Mode” for the first message and the word “address” after “Mode” for the second error message. **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 0x7fffff (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 (6018ms)
    withdraw
       ✓ should update cBalance based on compound exchange rate of cTokens (4805ms)
       ✓ 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)
    Liquidation when below 140% collateralization
       ✓ should request, lock, approve, withdraw, liquidate, accept, claim (6071ms)
    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,
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,
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 fundExpiry (709ms)
       ✓ should fail creating loan fund with > 10 years fundExpiry and maxLoanDur (187ms)
       ✓ should fail creating loan fund with 0 fundExpiry and maxLoanDur (276ms)
       ✓ should succeed in withdrawing from non-custom loan fund (3819ms)
       ✓ should succeed with setCompoundEnabled false if compoundSet is false (1580ms)
       ✓ should fail with setCompoundEnabled true 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 10 years+ fundExpiry and maxLoanDur (105ms)
       ✓ should fail creating custom loan fund with 0 fundExpiry and maxLoanDur (102ms)
       ✓ should succeed with setCompoundEnabled false if compoundSet is false (1464ms)
       ✓ should fail with setCompoundEnabled true if compoundSet is false (1114ms)
```

```
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
push funds
   ✓ should allow anyone to push funds to loan fund (383ms)
   ✓ should request and complete loan successfully if loan setup correctly (2524ms)
opening loan fund
   ✓ should increment fundIndex (292ms)
set fund details
   ✓ should allow changing of pubk (142ms)
   ✓ should allow changing of fund details (360ms)
   ✓ should fail changing of fund details with 10 years+ fundExpiry and maxLoanDur (119ms)
update
   ✓ should fail if not called by lender (278ms)
updateCustom
   ✓ 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 fail withdrawing funds if not called by owner (930ms)
   ✓ should allow withdrawing to a specific address as long as it's called by the owner of the fund (906ms)
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
   ✓ should convert 0 to MAX_LOAN_LENGTH + now if addNow bool is true (41ms)
setLoans
   ✓ should not allow setLoans to be called twice (94ms)
   ✓ should not allow setLoans to be called by address other than deployer (90ms)
   ✓ should fail if token is pausable and paused (1073ms)
setCompound
   ✓ should not allow setLoans 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 (90ms)
   ✓ should set maxUtilizationDelta if called by deployer (98ms)
setGlobalInterestRateNumerator
   ✓ should fail if set by non deployer (86ms)
   ✓ should set maxUtilizationDelta if called by deployer (106ms)
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 globalInterestRate if called by deployer (110ms)
setMinInterestRateNumerator
   ✓ should fail if set by non deployer (90ms)
   ✓ should set globalInterestRate if called by deployer (134ms)
setInterestUpdateDelay
   ✓ should fail if set by non deployer (92ms)
   ✓ should set globalInterestRate 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 (307ms)
   ✓ 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 (1366ms)

Contract: USDC Funds
 create
   ✓ should fail if user tries to create two loan funds (202ms)
   ✓ should succeed in updating non-custom loan fund (318ms)
   ✓ should fail in updating non-custom loan fund with > 10 years maxLoanDur (288ms)
   ✓ should fail in updating non-custom loan fund with > 10 years fundExpiry (271ms)
   ✓ should fail creating loan fund with > 10 years fundExpiry and maxLoanDur (102ms)
   ✓ should fail creating loan fund with 0 fundExpiry and maxLoanDur (92ms)
   ✓ should succeed in withdrawing from non-custom loan fund (1678ms)
   ✓ should succeed with setCompoundEnabled false if compoundSet is false (1332ms)
   ✓ should fail with setCompoundEnabled true if compoundSet is false (1162ms)
   ✓ should deposit funds on create if amount > 0 (741ms)
 createCustom
   ✓ should fail if user tries to create two loan funds (266ms)
   ✓ 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 with setCompoundEnabled false if compoundSet is false (1424ms)
   ✓ should fail with setCompoundEnabled true if compoundSet is false (1045ms)
 deposit
```

```
    ✓ 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)
updateCustom
    ✓ 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 (433ms)
    ✓ should fail if amount is greater than max loan amount (904ms)
    ✓ should succeed requesting from non-custom fund (6042ms)
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 (107ms)
    ✓ should not allow setLoans to be called by address other than deployer (83ms)
    ✓ should fail if token is pausable 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 (133ms)
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 (105ms)
    ✓ 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 (302ms)
    ✓ 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)
decreaseTotalBorrow
    ✓ should fail if msg.sender isn't lender (302ms)
  decreaseTotalBorrow
    ✓ should fail calling if not loans contract address (110ms)
    ✓ should fail decreaseTotalBorrow if totalBorrow is 0 (1388ms)

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 (2283ms)
  callSales
    ✓ 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 (12963ms)

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)
  setP2WSH
      ✓ should fail if msg.sender is not deployer (1302ms)
      ✓ should fail if p2wsh 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 (1088ms)
  unsetOnDemandSpv
      ✓ should fail if msg.sender not deployer (1921ms)
      ✓ should fail if onDemandSpv has not been set already (1219ms)
      ✓ should set onDemandSpv address to null (1257ms)
      ✓ 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 (1398ms)
      ✓ 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 (730ms)
      ✓ 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 (3155ms)
  refund
      ✓ should return loan repayment to borrower (2910ms)
      ✓ 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 (3096ms)
      ✓ 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 (1311ms)
      ✓ should succeed at creating a sale if below liquidation ratio (4131ms)
  default
      ✓ should fail liquidation 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 (94ms)
      ✓ should fail if secret provided does not hash to secretHashA1 (202ms)
```

```
    ✓ 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 depositd into loan (879ms)
    ✓ should return boolean determining whether funds have been depositd into loan (820ms)
  approved
    ✓ should return boolean determining whether loan has been approved (127ms)
  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 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 (1040ms)
    ✓ should fail if onDemandSpv already set (1634ms)
    ✓ 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)
    ✓ should allow loan to be created and repaid without onDemandSpv set (4949ms)
  setCollateral
    ✓ should fail if msg.sender is not deployer (1208ms)
    ✓ 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 (2660ms)
    ✓ 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 (3328ms)
    ✓ should accept successfully if arbiter secret provided (2788ms)
    ✓ 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 (2326ms)
    ✓ 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 (405ms)
    ✓ 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 (2803ms)
    ✓ 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 (1686ms)
    ✓ 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 (3263ms)
  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 pausable and paused (2873ms)
  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 (114ms)
    ✓ should return principal + interest + fee - repaid if parts of the loan were repaid (1723ms)
  funded
    ✓ should return boolean determining whether funds have been depositd into loan (1412ms)
    ✓ should return boolean determining whether funds have been depositd 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 should change upon partial repayment (1609ms)

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 (3003ms)
    ✓ 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 hasSecrets 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 providing signature for incorrect sale (93ms)
    ✓ should fail if msg.sender isn't borrower, lender, or arbiter (1256ms)
    ✓ should fail if current time is greater than settlementExpiration (1504ms)
    ✓ should fail if refundableSig is null (1649ms)
    ✓ should fail if seizableSig is null (1605ms)
  hasSecrets
    ✓ should return 0 if no secrets provided (5004ms)
  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 (2399ms)
    ✓ should fail refunding if discountBuy already accepted (3423ms)
    ✓ should fail if discountBuy is 0 (101ms)
  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 (1835ms)
  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 (3061ms)
    ✓ 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 (3980ms)
    ✓ should disperse funds to lender, arbiter, and medianizer if there is enough funds for owedToLender,
fee and penalty but not enough for borrower (2737ms)
    ✓ should disperse funds to rightful parties after partial repayment using provideSecretsAndAccept
function (3872ms)
    ✓ 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 (1390ms)
      ✓ should fail if current time is greater than settlementExpiration (1610ms)
      ✓ should fail if refundableSig is null (1578ms)
      ✓ should fail if seizableSig is null (1220ms)
    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 (503ms)
    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 (9053ms)
    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 account for delayed 6 conf proofs in collateral value (15650ms)
      ✓ should correctly update state if 6 conf proof comes before 1 conf proof (4813ms)
    Locking collateral multiple times
      ✓ should allow adding of temporary refundable collateral 20 times without running out of gas (40911ms)
    Adding collateral that is below 1% of
      ✓ should fail adding seizable collateral (7282ms)
      ✓ should fail adding refundable collateral (7103ms)
      ✓ 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 (9200ms)
    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)
    onDemandSpv service `paysValue`
      ✓ should be 1% of collateral value (3543ms)
    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 (4309ms)
      ✓ should return the refundable + seizable + temporary refundable + temporary seizable when adding
collateral in queue while satisfying both minSeizableCollateral and temporaryCollateral expiration (5805ms)
    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 not 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 (2443ms)
      ✓ 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

  21 passing (31ms)
```

Code Coverage

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

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

d72b5cf95b4c2d801c2ed373e26f2c22004dc3bb3dbaf9624a302384e6a39755 ./HotColdWallet.sol

f601587ed7d6ae4a3ffb2763903ec8637b43218b78aec193643759142780d07f ./FundsInterface.sol

a6313aa262893d039ae647614dd38bb7a1499b8262212688f8a3d48915b96368 ./WETH9.sol

9973ba9476caea4b2f875ad738637cca1ff420c20d1b2dc53831e1abbc3df16d ./ERC20.sol

```
b556566e7e11d60bc01e9f4f5194404068a3c26cf049f8cc0efb8fd5b5255f89 ./Medianizer.sol
04d2e9670d35b83540bf237d797ab01079c74a30aa46fe73ee2123acdc29aa44 ./ExampleDaiCoin.sol
294b63c0466a20d010adcacf24358da763974e917c0088d7d2c8fdd0b6567c37 ./Migrations.sol
2a8fc8b802902b65e6c7a4087b6c8876acf003b9f900dd6f64a84c1be2f57774 ./Oracle.sol
a2224011816430adaaa01575d259dbec5e9faceaf1a014d7e78749b3272f209b ./OracleInterface.sol
c248d6e912eb67b18e2a0d2671ace9b4cd5e847f7f6da270e16eb74401656cf3 ./MedianizerInterface.sol
e94514754ed4246037e285f1d8023d506ea8262479d154e97ba8d22f9aea51fc ./Buffer.sol
b08fc319b5f8ff47c0a85dabae1022b965175c5c33b9ba649a7c258fd9b61806 ./DSValue.sol
fecbb2f3c22dbf683cc7a2a84096e7621eb12e4dbe53cbe20306a5ef1c9ecea2 ./DSMath.sol
3ee490e702829d12f2f4319072210ed553be835c98f5e81818c66cbb58044b38 ./ExampleLinkCoin.sol
e2ce0a4fa04f32157c6d2db7db9b8e2c52ffd6ec7d92c59445361931217a16c8 ./WETH.sol
40bd477c3d2444658ee4326c2dc590cf246460512c16ef09344344e549eb54fd ./UniswapExchangeInterface.sol
41b603d872438bf42ffb74d752cc586e23ac0f9d9fbd1fd81baeb85dfc86ddc7 ./FundOracles.sol
fe7973ee69c8644c8f5549e2e4da54920bc2127f127ee40c5d8e5e02b5665aa6 ./Kraken.sol
b12214b154e59919691b44601a0353073dcb7b7cc6acfc7b050279b8431fb9c1 ./CryptoWatch.sol
8268136eb12a0a030127d0dd4662cd00e52da76edfdca0121ea7dccfb2113c8f ./OraclizeAPI.sol
41a4faf0e0aea0b68d80a49096df23333a2268e9b2d22de4a582cc7de14973d9 ./Bitstamp.sol
51fc75f8c18ee2b8e49b7ecad2d583d50e1256aa155a3485863117f5ec218cd2 ./Oraclize.sol
3b987cbdced3b6825e121ff0a665e5383f7c2c663009b3e3348541dadedd2b8a ./Coinpaprika.sol
591ee273a5dc78da55a766c9d0fa708d74ff9b5de571faee0f4c95d03b7baa58 ./a.sol
bc01cf98e192047018536783d5af13da3fd016ba8ec5395bc0334f825566e807 ./Coinbase.sol
4c08650b22a21c0a81ef4ce935262ad78ee46a218fec2f6b9b9948d60502bcea ./Chainlinked.sol
b38eb090ef9a9dd6b4d930ec2955ca7e00e04a7d75746ee904c638200947e3d7 ./BlockchainInfo.sol
d54065f8b0fa91f123cc02df10d437526b5219f3fbf0248499117aa442cc23a2 ./CryptoCompare.sol
061a76b8b1c7f3a956e3f773a0b5cb314a9ea0455df8a20259a22e4cb463e31f ./Gemini.sol
8c234ad0cd9a9966cbfbbe0f81dd2f3b8ef9f27bb7828cacf0a1d3bb92167384 ./CoinMarketCap.sol
323eab58630b41e0e086c31a1bfd2e13ef59b2373790a64bfab5e039ade6fa0e ./BitBay.sol
76842efe45175e264c7ff4574e987d061a11d4a8f8bbbfbe2e8e31fe26c756e5 ./ChainLink.sol
57094cedb1a69350c90468fef1ff29ff2ac8990805a3f89707bf0f63c8719042 ./ChainlinkedTesting.sol
77f014319bf8dcf5d6ba253c57d8428dd2ce88cd116513ac6308adfde321fb59 ./OraclizeAPITesting.sol
7a805a61562337c667b0eba24e465735261d525a94bbb77d9c8f5df788cb306b ./contracts/Sales.sol
ac0e52251e0cc10d81d5e6dfc392b274d99049cea0ea6fff9178a3cd66417c9d ./contracts/LoansInterface.sol
499afc6fc72021903d6d2abc0f07c55b341fd420a0f7a394cc429427fe0f697e ./contracts/MedianizerExample.sol
4a4ad171ea8485e3a166c6345006cf8a0bd6bef5a3c57790600a4f3f60334f9a ./contracts/P2WSHInterface.sol
acc82f42447eab93684c71fc15ae180bdc1db0ebe174dc27d5c4cc58c14bee71 ./contracts/Funds.sol
216bfcb92f9414e9b564b857837e46bc802b85198bf7c1346ee14a8d4f726758 ./contracts/ISPVRequestManager.sol
8b1cb2c78eba37b43c184ca5c6869641619a24f01ac014ab1292f1f7dde1490f ./contracts/ExampleUsdcCoin.sol
dcfa509e783170d145e08b5d758c39c78e405c5530f03d93207d8d70780c6f36 ./contracts/Medianizer.sol
1e5d54df59b0928751fb575f5411be46a9d18172e6dfc2778e39880250f0d8f3 ./contracts/ExampleDaiCoin.sol
c9e4224beadd6dba401f8becf3a4dcb0faa845342c7e2ff39cfce5fc87498375 ./contracts/Migrations.sol
57d43114a91148b542c527aa2bede0cac573bda17dac72508839bdd486fb8f1a ./contracts/Collateral.sol
cb75e2e0cb4a54ff9cb209cf02ab5de1cb7da843e329cc7d80a04a1b430417a0 ./contracts/ExamplePausableDaiCoin.sol
0c2da886f50281ce6724c68de9f2a068b84bab68649f3a8f480826618a887861 ./contracts/SalesInterface.sol
2f3cd052e088f8cabd0f9c3e4efb51ec16179df9ef80d1eba8cc22f1cf1dde76 ./contracts/Bytes.sol
de8e83778e67bc376dbc2de01a34d89169fb268a77bd03f0ff5e916816ca009f ./contracts/DSMath.sol
968897485b3632db4f89d301a0e3c60bbf8b20f3f37540af2d1cd6cac197e919 ./contracts/Loans.sol
ff262af55644e0cdc6b932a4fdbdc44a79a3d058967dc67eeb5a53168efe2f6d ./contracts/ALCompound.sol
b941f43e9283b2e16797a9a987c477b1908fdbb4ad2e751531ad5f5db13194d0 ./contracts/P2WSH.sol
d192b88a189676404b97b8f2d9846b531be7ab47e6bb7fbeac1e8c385f5f6650 ./contracts/ISPVInterfaces.sol
```

3a03535ae584f716ddf73db0ca8d801c9ba64d173f0b8add57012f85e326b7bc ./contracts/CollateralInterface.sol

e4888cf8c6a30f8c43cfbf8eb7b3b019d093c5ad83fe88c62e96fce1c59601b1 ./contracts/Compound/EIP20NonStandardInterface.sol

ea62b5e4cdfab2f5083c489fa60f3e6a11aa4fc41645721c9fb2f1b9cffe11d3 ./contracts/Compound/ComptrollerInterface.sol

6121f85e17166e591dc905f982931cf8e5c452221387176f502b44c1ef2853ca ./contracts/Compound/ETHInterestRateModel.sol

ccd95424bd137d67d5a84859671ae9a7726554414b00f2ad2bf47401b0df139d ./contracts/Compound/ReentrancyGuard.sol

357869a75a86e54fed1d7713cad7b15a7b21a9d494fa7b4b1cb62e126bc9f4cd ./contracts/Compound/Comptroller.sol

aff983be857e3645b0aef818f394d6fc892375920c1788846669173480a05e4a ./contracts/Compound/CErc20.sol

ecd4914563de59f27c5ca9a44e92cff2c7e9dd08e283702bd13d6cecf2695450 ./contracts/Compound/Exponential.sol

9075dbe3f4f95fb8c1931b89f7e2d4d0cf5a90d12c90f6bda7b9cfb928bdd664 ./contracts/Compound/PriceOracle.sol

98dad56878611705c2711a90c4b5fa9c230ef5e6873eddec1f122b973851a699 ./contracts/Compound/CToken.sol

daee15738b380ba17bbed81287577fb88e05cf5442147a1fb9b5fdf2654cea8e ./contracts/Compound/PriceOracleProxy.sol

149bc27a288b5eebe9474a47c8b71902fcb0b04048817846686e9aa8653e5810 ./contracts/Compound/ErrorReporter.sol

8b56d1c2b21a3fbc6d29b42bf0096d2ef347f48b9dcd2f52cecdcad72b2be6b8 ./contracts/Compound/Unitroller.sol

d42bbe588b09a848364fe1cd511346bd886872a75d760f91074af4287b9a0223 ./contracts/Compound/CarefulMath.sol

08993f91033c6eb492331d5981177f2089800016d92f12b40ce9397335dbc1b6 ./contracts/Compound/DAIInterestRateModel.sol

182b62e74a5cf212293620974756c46309c838dc9aa265b76f867b50c32126fa ./contracts/Compound/EIP20Interface.sol

32d07ba495299711e3aab55e732dff66bc77cd02032cc0f42579bc4e9dd19e03 ./contracts/Compound/InterestRateModel.sol

c4f32d62ceb5fe960ed36a77655638751cf36d9c01fc0dbfc59294f333eda308 ./contracts/Compound/ComptrollerStorage.sol

ebd3580e0755219dda5aa237073abf814c754edc4fad014adec2c6e0bc23ffb4 ./contracts/Compound/WhitePaperInterestRateModel.sol

97411113e0fdb2e0a04b26ec71a6b74a7fdf28fb66dbba79c97361f3e4f5c74e ./contracts/Compound/CEther.sol

a383b13ad79614dd4ce1b83b9020ad31e78c7373a71ce3f84fe1406bd3f703fb ./contracts/Compound/USDCInterestRateModel.sol

2e43a04996fc174ac0b3d3337d15c0e35dd92fc568ea34532145743da2ca362a ./contracts/Compound/SAIInterestRateModel.sol

a0903863f0949afb540fba18627462ab8eac8c407bae7a8d4bc366cd6785f723 ./contracts/Compound/Oracles/_ErrorReporter.sol

e4e0fe21f9b06b3d1d283a7afedcc91e6f87e5fccb0f4fece5728c9eb000fac6 ./contracts/Compound/Oracles/_Exponential.sol

3fe825c773b19cbb7434c84e1926ef403383cd8cc78c26513988ea6f568cd700 ./contracts/Compound/Oracles/_PriceOracle.sol

28b869531cdbe388ac355a816597f98fb1753ce29b1f9d8c4a6cb1d447f264aa ./contracts/Compound/Oracles/_DSValue.sol

8c72a4efc52b1f30ec59409c505b3a73c91ed43719755a9785a787f67a7ea936 ./contracts/Compound/Oracles/_MakerMedianizer.sol

3a99c6825cfbdb954de7cc31b54b2241f0d2f82854c875d21543a4e1aad5d209 ./contracts/Compound/Oracles/_CarefulMath.sol

Tests

26766745e6777e7ac5a00ec086a66d54edd6d73ca7c9654e1ebcd1f475f2a620 ./hotColdWallet.js

b49d35644e6fd7b07c349e49fdedc30edb327dbfda70ba706dd93ef3ba4c8aa6 ./mockJsonRpc.js

0ef23207589a6fdbdd918c87dcedd4e47d5e85393e6f05c44abd590fd3830213 ./rpc.js

fd2b4b4d5c9462a95c933d7ca65dc4de6cfefa1d3630141374b17252a08a39d7 ./rpc.js

4c956f4b7a7de72c672600394b910da090a36ce3c337476757829ad66044bb20 ./config.js

9b94259311e9be00afbd37baca1edb3817caa066170fa4dfe5bcaa1fa0b3fb34 ./common.js

9933b598550e1abf4056624352715ae977ff1b6cdde3575a0f431285ca4f2e21 ./liquidation.js

a3afbcb97cbde69bcb66fc83b7433174fb2fe33eabb1c06944a442ae3def7bbd ./collateral.js

edb8edfbe0600f0b301ee2aaae8d2937ce02ac986e2eacaf0760eb9cdc3af738 ./collateralSwap.js

111a834765a7b78454cb8aad10c8e64e46bd3fda7946f86a7795c093c5fd5668 ./transactions.js

9a8be2544e94a558930d29a6319728682fb4bfdfe42dd563e858ec82d2f3a91d ./chainlink.js

55906a60054e348ed731f6518183785c8dc02fa8b59ed2e07d4888c0f581fd15 ./medianizer.js

82e0a6fc152fdf8769dbbbbeb9cc8cbafa8e88ba3ebe180872d2bde3227ccc77 ./oraclize.js

48ebad634566ebeba418bc6d6960655fbe3bb4714b6203eaa35aa6be9e567534 ./Utils.js

531fb9f48b90ccecb610102c64ae71f541874abe1fa4f279b8f936cca264f695 ./test/sales.js

2f5c9fd3ec04114295753d1596b5e986ed7c0da4b89d28f24f20ebc7b9a8ae37 ./test/spv.js

7cd18be85405bda29f5defa60eec6289f808005e512c86d7fc2b543be513e284 ./test/interestIncrease.js

90fc7d30370a2d997eeb48a35f6b8fb6ed13540dc808f3a67c57597e1f6f7cee ./test/interestDecrease.js

3c0ec87806a0b8c266ba7e6dbdc91fefe20398282c420d3cd21f713b8a1c9545 ./test/p2wsh.js

078ac197cdb975cfe47a05e09207c1551e0a874b81e1895c015e2d71c02b265e ./test/bytes.js

1d82baa5a1733cf0bc655c6e94be240a580c9e959a029ab343c611cb6aba76a6 ./test/loans.js

51c65c8fdd615285e2fc156c1651889648a217cdcde96646e3ccea3c66dfa3b9 ./test/compound.js

4037f1f7816f49b5a838681d21f6d2b4ee93588c2651cf1eefb1b9442db5fe28 ./test/e2e.js

439f0d3fca2fb6d4f26274de0ace92f35997cc72a8d163fb708ae4f5f9dabaf0 ./test/alcompound.js

3666f553fa4eb87ab1bc931f6e7a8389cd3c81ea4ccd32087d49f2052176417f ./test/funds.js

48ebad634566ebeba418bc6d6960655fbe3bb4714b6203eaa35aa6be9e567534 ./test/helpers/Utils.js

5335ccae542710594639bc2a5e8412c7283dab14850fe691a34aef9b32491e30 ./test/helpers/collateral/config.js

36de08bf23e42dc05228660d5d705d159ef783d095b4acf14c150fd08515a9a3 ./test/helpers/collateral/common.js

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.