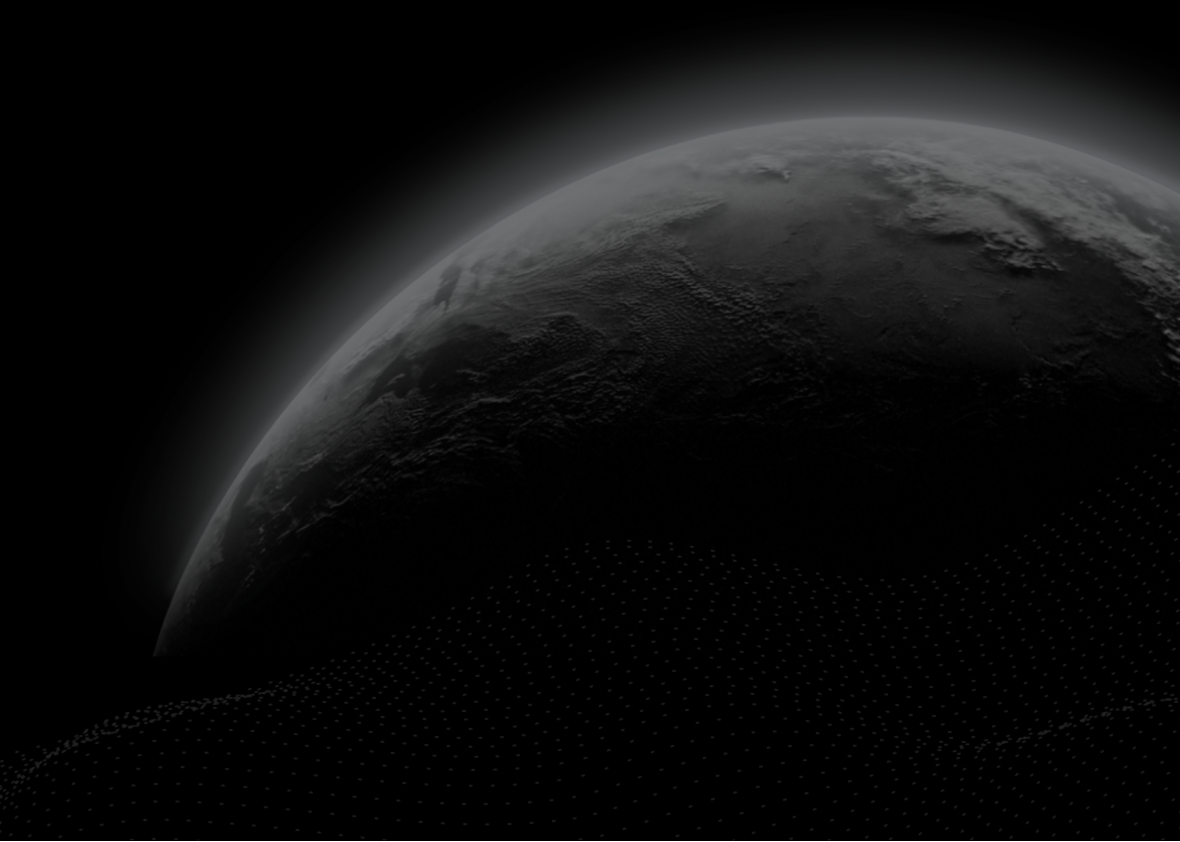
CERTIK

Security Assessment

Gondi (Addendum 2)

CertiK Assessed on Jul 27th, 2023



CertiK Assessed on Jul 27th, 2023

Gondi (Addendum 2)

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Lending	Ethereum (ETH)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 07/27/2023	N/A

CODEBASE

changes introduced by commit excluding `test` folder, `src/lib/loans` was fully audited

changes introduced by commit excluding

View All in Codebase Page

COMMITS

13f392689d0ec59dab2f7e4190c34f532de9d946

918dcc63e660f57722fbb6b407a90152449770bf

View All in Codebase Page

Vulnerability Summary

19	19	0	0	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥 0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧 0	Major		Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨 6	Medium	6 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 5	Minor	5 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦 8	Informational	8 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | GONDI (ADDENDUM 2)

Summary

Findings

Appendix

Disclaimer

CODEBASE | GONDI (ADDENDUM 2)

Repository

changes introduced by commit excluding `test` folder, `src/lib/loans` was fully audited

changes introduced by commit excluding `AuctionLoanLiquidator.sol`

Commit

13f392689d0ec59dab2f7e4190c34f532de9d946 918dcc63e660f57722fbb6b407a90152449770bf

AUDIT SCOPE | GONDI (ADDENDUM 2)

5 files audited ● 3 files with Resolved findings ○ 2 files without findings

ID	Repo	File	SHA256 Checksum
● BLB	pixeldaogg/florida-contracts	src/lib/loans/BaseLoan.sol	b980221e40eb328966b4756936ee0f4152a2b39ccba90f3369760bf6675a8429
● MSL	pixeldaogg/florida-contracts	src/lib/loans/MultiSourceLoan.sol	6344ef6b577daa5ff19e124ef31cc0e162752a621fff0bee326f587f384bdd90
● SSL	pixeldaogg/florida-contracts	src/lib/loans/SingleSourceLoan.sol	7f65613484924745fb416a3f5e451a7e11251ce582cc5b93288337f170db4810
○ BLU	pixeldaogg/florida-contracts	src/lib/loans/BaseLoan.sol	a2177ceddfabf21a84bf39b9721786f7f19dc2a8957b90736a19b4b289f01624
○ MUL	pixeldaogg/florida-contracts	src/lib/loans/MultiSourceLoan.sol	addc69509750729d897ce90b07b8eabf949467f81c09f8c65ad17d6608073160

APPROACH & METHODS | GONDI (ADDENDUM 2)

This report has been prepared for Gondi to discover issues and vulnerabilities in the source code of the Gondi (Addendum 2) project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | GONDI (ADDENDUM 2)

19	0	0	6	5	8
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Gondi (Addendum 2). Through this audit, we have uncovered 19 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
BLB-01	`cancelRenegotiationOffers()` Cancels Normal Offers	Inconsistency	Medium	● Resolved
BLB-02	Wrong `LOAN_MANAGER_ID`	Inconsistency	Medium	● Resolved
MSL-01	In `MultiSourceLoan._baseRenegotiationChecks()` It Is Not Checked That The Offer Is Not Cancelled	Volatile Code	Medium	● Resolved
MSL-03	Invalid `_checkStrictlyBetter()` Arguments In `MultiSourceLoan.refinanceFull()`	Volatile Code	Medium	● Resolved
MSL-04	Wrong Handling Of `_refinanceOffer.fee` In `_refinancePartial()`	Incorrect Calculation	Medium	● Resolved
MSL-05	Different Usage Of `_minimum.interest` In `_processOldSources()`	Inconsistency	Medium	● Resolved
BLB-04	Function State Mutability Can Be Restricted To `view`	Inconsistency	Minor	● Resolved
LOA-01	Missing Zero Address Validation	Volatile Code	Minor	● Resolved
MSL-06	`_refinanceOffer.signer` Is Not Checked	Volatile Code	Minor	● Resolved
SSL-01	Wrong `_transferredIn` Passed To `validateLoan()` In `renegotiateLoan()`	Volatile Code	Minor	● Resolved
SSL-02	Unsafe Operations In Loan Liquidation Workflow	Volatile Code	Minor	● Resolved

ID	Title	Category	Severity	Status
BLB-05	`BaseLoan.cancelAllOffers()` Can Be Executed Twice	Coding Issue	Informational	● Resolved
LIB-01	`_tokenId` Is Supposed To Be `_loanId`	Coding Style	Informational	● Resolved
LOA-02	Protocol Fee Is Not Taken In `emitLoan()`	Inconsistency	Informational	● Resolved
LOA-03	`LoanNotFoundError` Is Misleading	Coding Style	Informational	● Resolved
LON-01	Inaccurate Comments	Coding Style	Informational	● Resolved
MSL-07	`withProtocolFee` Is Not Checked In `MultiSourceLoan.repayLoan()`	Volatile Code	Informational	● Resolved
SRC-01	Unused Declarations	Inconsistency	Informational	● Resolved
SSL-03	`+=` Can Be Used	Coding Style	Informational	● Resolved

BLB-01 | `cancelRenegotiationOffers()` CANCELS NORMAL OFFERS

Category	Severity	Location	Status
Inconsistency	● Medium	src/lib/loans/BaseLoan.sol (base): 377	● Resolved

Description

```
377            isOfferCancelled[_lender][renegotiationId] = true;
```

`isRenegotiationOfferCancelled` is supposed to be updated in `cancelRenegotiationOffers()` .

Recommendation

We recommend updating `isRenegotiationOfferCancelled` or using common numbering of normal and renegotiation offers.

BLB-02 | WRONG `LOAN_MANAGER_ID`

Category	Severity	Location	Status
Inconsistency	● Medium	src/lib/loans/BaseLoan.sol (base): 699	● Resolved

Description

The contract `LoanManagerId` declares `LOAN_MANAGER_ID = 0x863af7bc` . The value is misleading since

```
LoanManager.onLoanRepaid.selector = 0xade3a41e
LoanManager.validateLoan.selector = 0x99e67b8e
and
type(ILoanManager).interfaceId = LoanManager.onLoanRepaid.selector ^
LoanManager.validateLoan.selector = 0x3405df90
assuming ILoanManager declares two functions
```

Recommendation

We recommend clarifying the origin of the value or using the proposed methods of calculation.

MSL-01 | IN `MultiSourceLoan._baseRenegotiationChecks()` IT IS NOT CHECKED THAT THE OFFER IS NOT CANCELLED

Category	Severity	Location	Status
Volatile Code	● Medium	src/lib/loans/MultiSourceLoan.sol (base): 779	● Resolved

Description

`BaseLoan` defines `isRenegotiationOfferCancelled` / `lenderMinRenegotiationOfferId` , however, they are not checked in `MultiSourceLoan._baseRenegotiationChecks()` . This disallows `_refinanceOffer` to be cancelled by the lender.

Recommendation

We recommend checking if `_refinanceOffer` is cancelled.

MSL-03 | INVALID `_checkStrictlyBetter()` ARGUMENTS IN `MultiSourceLoan.refinanceFull()`

Category	Severity	Location	Status
Volatile Code	● Medium	src/lib/loans/MultiSourceLoan.sol (base): 200~208	● Resolved

Description

```
200             _checkStrictlyBetter(
201                 _refinanceOffer.principalAmount,
202                 totalDelta,
203                 _refinanceOffer.duration,
204                 currentDuration,
205                 _refinanceOffer.aprBps * _refinanceOffer.principalAmount,
206                 totalAnnualInterest,
207                 _refinanceOffer.fee,
208                 _loan.startTime
```

The second argument is expected to be the old principal. The new principal is expected to be 1% lower than the old one (with default `_minimum`). However, `totalDelta` is passed, that is the amount repaid by the refinance lender, not the old principal. Passing the "strictly better" condition is significantly easier.

Arguments 5 and 6 are expected to be new and old `aprBps` , however, annual interests are passed instead. As a result, instead of `aprOld * principalOld - aprNew * principalNew` it will be calculated `aprOld * principalOld * principalOld - aprNew * principalNew * principalNew` . Interest delta is expected to be at least 1% of the old interest. This also makes it easier to pass "strictly better" condition. For example, halving the principal should give a 50% improvement, but gives 75%.

Recommendation

We recommend using the same checks for single and multi source loans.

MSL-04 WRONG HANDLING OF `_refinanceOffer.fee` IN `_refinancePartial()`

Category	Severity	Location	Status
Incorrect Calculation	● Medium	src/lib/loans/MultiSourceLoan.sol (base): 596	● Resolved

Description

`MultiSourceLoan._refinancePartial()` works this way:

1. New lender prepares `_refinanceOffer` and calls `refinancePartial()` / `refinancePartialBatch()`
2. `_processOldSources()` calculates `totalDelta`
3. `_processOldSource()` transfers the `delta` with interest from new to each old lender
4. It is ensured `totalDelta == _refinanceOffer.principalAmount`
5. If lender is a vault, `validateLoan()` is called with `_transferredIn = _refinanceOffer.fee`

However, the fee was not taken by new lender, they covered fully `totalDelta` and accrued interest.

Recommendation

We recommend ensuring `totalDelta == _refinanceOffer.principalAmount - _refinanceOffer.fee` instead.

MSL-05 | DIFFERENT USAGE OF `_minimum.interest` IN `_processOldSources()`

Category	Severity	Location	Status
Inconsistency	● Medium	src/lib/loans/MultiSourceLoan.sol (base): 658~660	● Resolved

Description

```
655             if (
656                 _isStrictlyBetter &&
657                 delta > 0 &&
658                 ((source.aprBps - _refinanceOffer.aprBps).mulDivDown(
659                     _PRECISION,
660                     source.aprBps
661                 ) < _minimum.interest)
662             ) {
663                 revert InvalidRenegotiationOfferError();
664             }
```

`_minimum.interest` is supposed to set minimal interest improvement for `_isStrictlyBetter` offers. However, `_processOldSources()` checks if `aprBps` is improved by this value instead.

For example, if `_targetPrincipal` is half of `_source.principalAmount` and `aprBps` is the same, the interest is halved and should be "strictly better", but the transaction is reverted with `InvalidRenegotiationOfferError` .

Recommendation

We recommend clarifying the intended behavior.

Alleviation

The project team confirmed the behavior is intended.

BLB-04 | FUNCTION STATE MUTABILITY CAN BE RESTRICTED TO `view`

Category	Severity	Location	Status
Inconsistency	● Minor	src/lib/loans/BaseLoan.sol (base): 486	● Resolved

Description

`BaseLoan.getLiquidationAuctionDuration()` state mutability can be restricted to `view` . The function is supposed to be called off-chain.

Recommendation

We recommend using `view` modifier.

LOA-01 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	src/lib/loans/BaseLoan.sol (base): 223~225, 267, 456, 471; src/lib/loans/MultiSourceLoan.sol (base): 80, 362; src/lib/loans/SingleSourceLoan.sol (base): 73, 210	● Resolved

Description

The cited address input is missing a check that it is not `address(0)` .

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors.

MSL-06 | `_refinanceOffer.signer` IS NOT CHECKED

Category	Severity	Location	Status
Volatile Code	● Minor	src/lib/loans/MultiSourceLoan.sol (base): 191	● Resolved

Description

`_refinanceOffer.signer` field is not checked in `MultiSourceLoan.refinanceFull()` in case of `strictImprovement`.

Recommendation

We recommend ensuring the field is zero despite the fact it is not used.

Alleviation

Since the field is unused in the mentioned scenario, the finding is marked as Resolved.

SSL-01 | WRONG `_transferredIn` PASSED TO `validateLoan()` IN `renegotiateLoan()`

Category	Severity	Location	Status
Volatile Code	● Minor	src/lib/loans/SingleSourceLoan.sol (base): 411	● Resolved

Description

```
407         if (_vaultDirectory.vaultExists(_renegotiationOffer.lender)) {
408             LoanManager(_renegotiationOffer.lender).validateLoan(
409                 newLoanId,
410                 _renegotiationOffer.principalAmount + accruedInterest,
411                 _renegotiationOffer.fee,
412                 abi.encode(_loan)
413             );
414         }
```

In `SingleSourceLoan.renegotiateLoan()` if the new lender is a vault, it is informed about the incoming amount via the call to `validateLoan()`. `_renegotiationOffer.fee` is passed as `_transferredIn` argument. However, the amount transferred in reality is lower by `protocolFeeFromFee`. This can lead to a wrong bookkeeping in `Vault._processLoanIncome()`.

Recommendation

We recommend passing the real amount transferred to Vault.

SSL-02 | UNSAFE OPERATIONS IN LOAN LIQUIDATION WORKFLOW

Category	Severity	Location	Status
Volatile Code	● Minor	src/lib/loans/SingleSourceLoan.sol (base): 526	● Resolved

Description

The loan liquidation works this way:

1. Lender or their signer calls `liquidateLoan()`
2. If `_loan.requiresLiquidation` , `_loanLiquidator.liquidateLoan()` is called
3. Auction lasts for `_liquidationAuctionDuration`
4. Someone calls `AuctionLoanLiquidator.settleAuction()`
5. `auction.highestBid` is transferred to `loanAddress`
6. `loanAddress.loanLiquidated()` is called
7. In `loanLiquidated()` `highestBid` is transferred to `lender`
8. Loan is deleted
9. Auction is deleted

This workflow relies on the implementation details of other parts

1. It is better to `approve(loanAddress, highestBid)` in `settleAuction()` instead of transferring. This will make sure that the `loanAddress` will only spend the tokens from the auction, never its own. In the current implementation `AuctionLoanLiquidator` can forget to transfer funds.
2. It is better to mark the `_loanId` as being liquidated in `liquidateLoan()` as soon as the liquidation process starts. The current implementation relies on the `nonReentrant` modifier in `AuctionLoanLiquidator` . See the scenario section.

Scenario

This scenario currently can't be executed due to the `nonReentrant` modifier in `AuctionLoanLiquidator` , however, it demonstrates the potential issues.

1. Lender calls `loanContract.liquidateLoan()`
2. `_loanLiquidator.liquidateLoan()` is called, `_loans[_loanId]` is kept active
3. Auction lasts for `_liquidationAuctionDuration` , the lender takes part and raises bids to influence the final price
4. If the lender accidentally wins the auction they call `_loanLiquidator.settleAuction()`
5. When the collateral is transferred to the lender, the `onERC721Received()` hook is called and the lender gets control

6. In the same transaction lender transfers the collateral back to the `loanContract`
7. In the same transaction lender calls `loanContract.liquidateLoan()` **again** since `_loans[_loanId]` is still active.

Recommendation

We do not recommend relying on the implementation details of other contracts even if they are part of the project.

BLB-05 | `BaseLoan.cancelAllOffers()` CAN BE EXECUTED TWICE

Category	Severity	Location	Status
Coding Issue	● Informational	src/lib/loans/BaseLoan.sol (base): 347, 397	● Resolved

Description

```
347         if (currentMinOfferId > _minOfferId) {
348             revert LowOfferIdError(_lender, _minOfferId, currentMinOfferId);
```

Calling the function with `_minOfferId` equal to `currentMinOfferId` will emit the event `AllOffersCancelled` .

Recommendation

We recommend checking `currentMinOfferId >= _minOfferId` to avoid unnecessary execution.

LIB-01 | `_tokenId` IS SUPPOSED TO BE `_loanId`

Category	Severity	Location	Status
Coding Style	● Informational	src/lib/Vault.sol (base): 626; src/lib/loans/BaseLoan.sol (base): 716, 730	● Resolved

Description

The `_tokenId` argument of `Vault.validateLoan()` and `onLoanRepaid()` is supposed to be `_loanId` .

Recommendation

We recommend renaming the argument.

LOA-02 | PROTOCOL FEE IS NOT TAKEN IN `emitLoan()`

Category	Severity	Location	Status
Inconsistency	● Informational	src/lib/loans/MultiSourceLoan.sol (base): 140, 222; src/lib/loans/SingleSourceLoan.sol (base): 131	● Resolved

Description

In `emitLoan()` the borrower gets `_loanOffer.principalAmount - _loanOffer.fee` , however, the `_protocolFee.recipient` doesn't get the `_protocolFee.fraction` of fee.

`_renegotiationOffer.fee` and accrued interest are taxed by `_protocolFee.fraction` in `SingleSourceLoan.renegotiateLoan()` .

`_refinanceOffer.fee` and accrued interest are not taxed by `_protocolFee.fraction` in `MultiSourceLoan.refinanceFull()` .

Recommendation

We recommend clarifying the intended behavior.

LOA-03 | `LoanNotFoundError` IS MISLEADING

Category	Severity	Location	Status
Coding Style	● Informational	src/lib/loans/MultiSourceLoan.sol (base): 422~424; src/lib/loans/SingleSourceLoan.sol (base): 160~165, 244~245, 503~505	● Resolved

Description

```
160         if (_loan.hash() != _loans[_loanId]) {
161             revert InvalidLoanError(_loanId);
162         }
163         if (_loan.borrower == address(0)) {
164             revert LoanNotFoundError(_loanId);
165         }
```

The first check ensures that the `_loanId` with the same content as `_loan` was created in `emitLoan()` and not yet liquidated/repaid.

The second check ensures that the loan previously created has a valid `borrower`. However, that is always true. The check is redundant and misleading.

When the loan is liquidated or repaid, its hash is deleted from `_loans`.

Recommendation

We recommend removing `LoanNotFoundError` or clarifying the intended behavior.

LON-01 | INACCURATE COMMENTS

Category	Severity	Location	Status
Coding Style	● Informational	src/interfaces/loans/IBaseLoan.sol (base): 105, 111, 117; src/interfaces/loans/IMultiSourceLoan.sol (base): 75; src/interfaces/loans/ISingleSourceLoan.sol (base): 12~19	● Resolved

Description

Some comments are inaccurate

- `_offerId` is supposed to be `_renegotiationId`
- `_offerIds` is supposed to be `_renegotiationIds`

Recommendation

We recommend updating the comments.

MSL-07 | `withProtocolFee` IS NOT CHECKED IN `MultiSourceLoan.repayLoan()`

Category	Severity	Location	Status
Volatile Code	● Informational	src/lib/loans/MultiSourceLoan.sol (base): 356	● Resolved

Description

In `MultiSourceLoan.repayLoan()` the `totalProtocolFee` is only accumulated if `withProtocolFee`, however, it is always transferred to `protocolFee.recipient`.

Recommendation

We recommend checking if `withProtocolFee` before transferring for consistency with other code.

SRC-01 | UNUSED DECLARATIONS

Category	Severity	Location	Status
Inconsistency	● Informational	src/interfaces/loans/IBaseLoan.sol (base): 10~14; src/lib/AuctionLoanLiquidator.sol (base): 121; src/lib/loans/BaseLoan.sol (base): 716~719; src/lib/loans/MultiSourceLoan.sol (base): 23, 555; src/lib/loans/SingleSourceLoan.sol (base): 27	● Resolved

Description

- `BaseLoan.onLoanRepaid()` doesn't use the declared arguments.
- `totalAnnualInterest` in `MultiSourceLoan._refinancePartial()` is never used.

The compiler will produce warnings.

- `LoanStatus` in `IBaseLoan` is never used.
- `_addLoanContract` in `AuctionLoanLiquidator` is never used.
- `LoanManagerId` inherited by `SingleSourceLoan` is never used. `LoanManager.onLoanRepaid.selector` is used directly.
- `MultiSourceLoan.liquidationAuctionDuration` can be replaced with configurable `BaseLoan._liquidationAuctionDuration` . It can also be declared `immutable.

Recommendation

We recommend removing of unused declarations.

SSL-03 | `+=` CAN BE USED

Category	Severity	Location	Status
Coding Style	● Informational	src/lib/loans/SingleSourceLoan.sol (base): 107~109	● Resolved

Description

```
107             _used[_loanOffer.lender][_loanOffer.offerId] =
108                 _used[_loanOffer.lender][_loanOffer.offerId] +
109                 _loanOffer.principalAmount;
```

`+=` operation can be used to improve readability.

Recommendation

We recommend using `+=` wherever possible.

APPENDIX | GONDI (ADDENDUM 2)

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.