CERTIK

Fetch.ai

Atomix Smart Contracts

Security Assessment

February 12th, 2021

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	Fetch.ai - Atomix Smart Contracts
Description	Smart contracts of the atomix_contracts repository.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. `707cca61374923246436f990447aae68570d6905` 2. `294675db10f0aeffb7ef442f1a6e320afa3599ed`

Audit Summary

Delivery Date	Feb. 12, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Feb. 1, 2021 - Feb. 6, 2021

Vulnerability Summary

Total Issues	39 (35 Resolved, 4 Informational Acknowledged)
● **Total Critical**	4 *(4 Resolved)*
● **Total Major**	0
● **Total Medium**	8 *(8 Resolved)*
● **Total Minor**	9 *(9 Resolved)*
● **Total Informational**	18 *(14 Resolved, 4 Acknowledged)*

Executive Summary

The codebase of Fetch.ai's Atomix repository was found to be well-written, but contained some inefficient usage of named return variables and function visibilities. Multiple cases were identified in the `LendingPool` contract where minor re-entrancy was possible, leading to events being emitted out of order but not compromising the state of the pool itself. In the same locations within the `LendingPool` contract, the `ERC20.transferFrom` function was called often, without checking its result. Not all ERC-20 implementations are guaranteed to revert, so we recommended to import the OpenZeppelin `SafeERC20` library and use its `safeTransferFrom` function instead.

While not in the scope of the audit, we noted that the `AtomixBase` contract declares two public virtual functions `onRegistryUpdate` and `onRegistryPostUpdate`, both of which take an `IContractRegistry` parameter and have no modifiers or requirements within their function bodies. Due to the manner of implementation, the `InterestManager`, `LendingPool` and `LendingPoolStorageModifier` contracts override these functions in order to apply changes to their state variables, taking the values from the supplied `IContractRegistry` parameter, before calling the base function implementation by way of `super`. No requirements or any form of access restriction is implemented in these functions, which allowed anyone to call them and supply their own `IContractRegistry` value, setting the state variables within each of the contracts to any of the values that they require. Additionally, the system became paused. We pointed out that this can be resolved by either introducing access restriction to the `onRegistryUpdate` and `onRegistryPostUpdate` functions, or by changing their visibility to internal in order to prevent external calling altogether. The corresponding functions are now declared as internal.

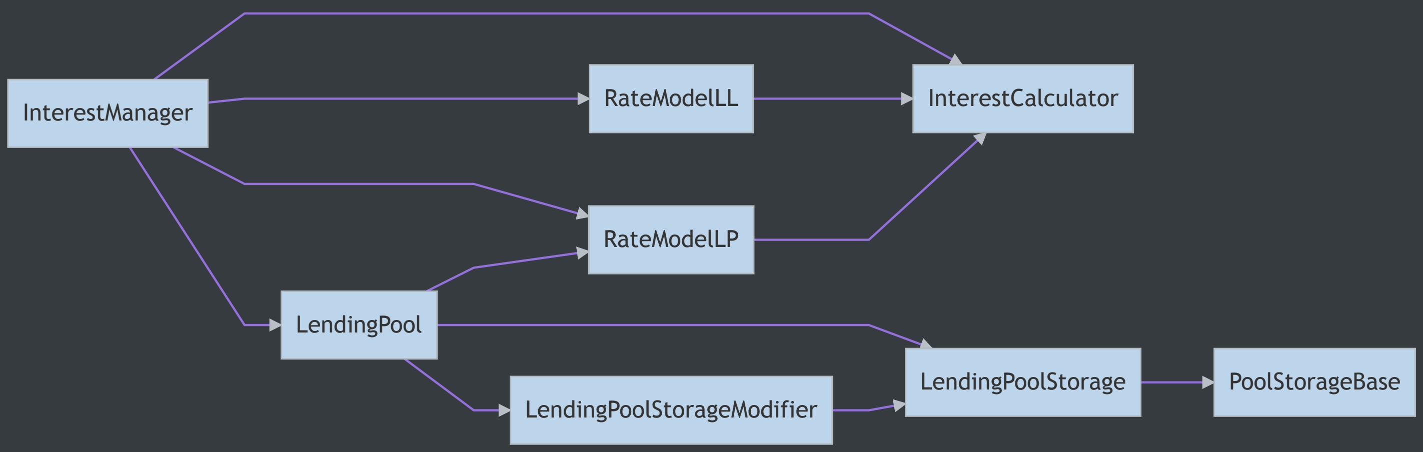
INC-04, LPO-19 and PSB-02 suggest refactoring functions into modifiers, which the Fetch.ai team agreed could be done, but stated that they seem to be more of a question of style. They have chosen not to create their own modifiers because the development environment, Brownie, has a bug where the code coverage tools do not instrument them properly. Implementing this as functions is no worse from a gas-cost POV and in fact, if comparing their method to a straight replacement with modifiers, their approach is more gas-efficient as the file size is smaller due to the code being implemented as a function rather than inlined.

During the course of the engagement, the Fetch.ai team have also been made aware of a re-entrancy issue with the `withdrawACT` which was consider of critical severity. The `msg.sender` (usually the borrower) could be a contract and implement the `ERC1155Receiver` function `onERC1155Received` and use it to borrow funds. This would have resulted in a borrower taking out a loan with no ACT collateral backing it. All of the specific re-entrancy issues have been resolved, and additional more general guards are in place for unseen attacks. The implementation was found to be implemented correctly. Key changes here:

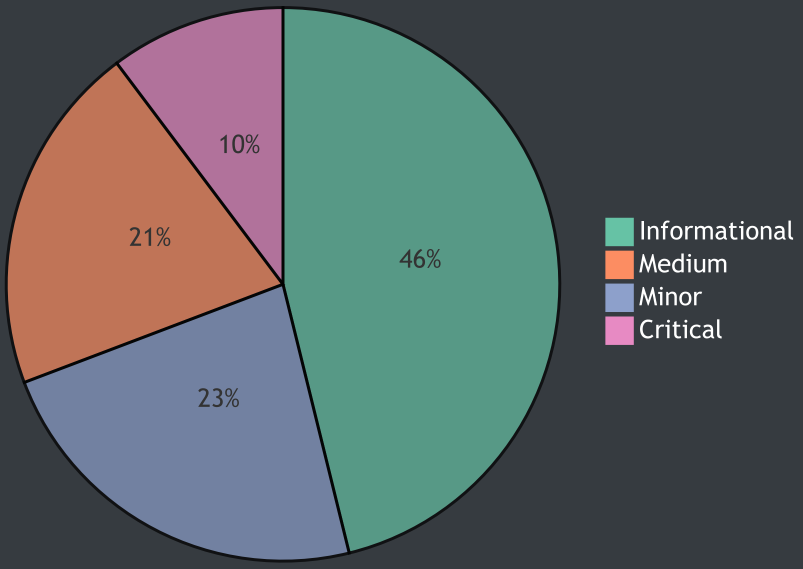
1. Require that `spreadDestinationWallet`, `lendingPoolWalletAddress` and `breachAddress` all implement the `AtomixWallet` interface (so we can be reasonably sure we are not passing in an incorrect contract when we deploy the system).
2. Relevant contracts implement the checks-effects-interactions pattern as well as inheriting from OpenZeppelin `ReentrancyGuard` and employing the `nonReentrant()` modifier.
3. There is an additional check at the end of `withdrawACT()` and `borrow()` to confirm that the borrower is within their borrowing limit when we exit the function.

Files In Scope

ID	Contract	Location
INC	InterestCalculator	contracts/ALP/InterestCalculator.sol
INM	InterestManager	contracts/ALP/InterestManager.sol
LPO	LendingPool	contracts/ALP/LendingPool.sol
LPS	LendingPoolStorage	contracts/ALP/LendingPoolStorage.sol
LSM	LendingPoolStorageModifier	contracts/ALP/LendingPoolStorageModifier.sol
PSB	PoolStorageBase	contracts/ALP/PoolStorageBase.sol
RLL	RateModelLL	contracts/ALP/RateModelLL.sol
RLP	RateModelLP	contracts/ALP/RateModelLP.sol



Findings



ID	Title	Type	Severity	Resolved
INC-01	Constant variables not following naming conventions	Naming Conventions	● Informational	✓
INC-02	Functions should be re-declared as external	Gas Optimization	● Informational	✓
INC-03	Redundant array length calculation	Gas Optimization	● Informational	✓
INC-04	Function should be refactored into a modifier	Implementation	● Informational	(!)

INC-05	Contradictory requirement	Volatile Code	● Medium	✓
INC-06	Redundant calculation	Arithmetic	● Informational	✓
INC-07	Unused named return variables	Implementation	● Informational	✓
INC-08	Potential integer truncation	Arithmetic	● Medium	✓
INM-01	Unused named return variables	Implementation	● Informational	✓
INM-02	Functions should be re-declared as external	Gas Optimization	● Informational	✓
INM-03	Unused named return variables	Implementation	● Informational	✓
INM-04	Unused named return variable	Implementation	● Informational	✓
INM-05	Lack of access restriction allows overriding state variables	Volatile Code	● Critical	✓
INM-06	Lack of access restriction allows overriding state variables	Volatile Code	● Critical	✓
LPO-01	Unnecessary usage of SafeMath functionality	Implementation	● Informational	✓
LPO-02	Functions should be re-declared as external	Gas Optimization	● Informational	✓
LPO-03	Potential integer underflow	Arithmetic	● Minor	✓

LPO-04	Unused named return variable	Implementation	🟢 Informational	✓
LPO-05	Unused result from call to transferFrom	Volatile Code	🟡 Medium	✓
LPO-06	Potential for minor re-entrancy; Out-of-order events	Volatile Code	🔵 Minor	✓
LPO-07	Potential for minor re-entrancy; Out-of-order events	Volatile Code	🔵 Minor	✓
LPO-08	Potential for minor re-entrancy; Out-of-order events	Volatile Code	🔵 Minor	✓
LPO-09	Unused result from call to transferFrom	Volatile Code	🟡 Medium	✓
LPO-10	Potential for minor re-entrancy; Out-of-order events	Volatile Code	🔵 Minor	✓
LPO-11	Unused result from call to transferFrom	Volatile Code	🟡 Medium	✓
LPO-12	Potential for minor re-entrancy; Out-of-order events	Volatile Code	🔵 Minor	✓
LPO-13	Unused result from call to transferFrom	Volatile Code	🟡 Medium	✓
LPO-14	Potential for minor re-entrancy; Out-of-order events	Volatile Code	🔵 Minor	✓
LPO-15	Unused result from call to	Volatile Code	🟡 Medium	✓

	transferFrom			
LPO-16	Potential for minor re-entrancy; Out-of-order events	Volatile Code	🔵 Minor	✓
LPO-17	Unused result from call to transferFrom	Volatile Code	🟡 Medium	✓
LPO-18	Potential for minor re-entrancy; Out-of-order events	Volatile Code	🔵 Minor	✓
LPO-19	Function should be refactored into a modifier	Implementation	🟢 Informational	ⓘ
LPO-20	Lack of access restriction allows overriding state variables	Volatile Code	🔴 Critical	✓
LSM-01	Lack of access restriction allows overriding state variables	Volatile Code	🔴 Critical	✓
LSM-02	Function should be refactored into a modifier	Implementation	🟢 Informational	ⓘ
PSB-01	Function should be re-declared as external	Gas Optimization	🟢 Informational	✓
PSB-02	Function should be refactored into a modifier	Implementation	🟢 Informational	ⓘ
RLP-01	Unused named return variables	Implementation	🟢 Informational	✓

INC-01: Constant variables not following naming conventions

Type	Severity	Location
Naming Conventions	● Informational	contracts/ALP/InterestCalculator.sol L13, L20-L25, L58

Description:

The `secsPerYear` , `a0` , `a1` , `a2` , `a3` , `a4` , `a5` and `numBins` constant variables in the `InterestCalculator` contract are not named in upper-case with underscores, which goes against the recommended Solidity naming conventions.

Recommendation:

Consider renaming the constant variables to `SECS_PER_YEAR` , `A0` , `A1` , `A2` , `A3` , `A4` , `A5` and `NUM_BINS` respectively.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INC-02: Functions should be re-declared as external

Type	Severity	Location
Gas Optimization	● Informational	contracts/ALP/InterestCalculator.sol L74, L144, L158, L165

Description:

The public `uploadRateData` , `getBorrowerAPRRate` , `minBorrowingApr` and `maxBorrowingApr` functions in the `InterestCalculator` contract is should be re-declared as external.

Recommendation:

Consider re-declaring the public `uploadRateData` function as external.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INC-03: Redundant array length calculation

Type	Severity	Location
Gas Optimization	● Informational	contracts/ALP/InterestCalculator.sol L76-L77

Description:

The public `uploadRateData` function in the `InterestCalculator` contract queries the length of the supplied `rateData` array parameter multiple times, which is inefficient.

Recommendation:

Consider storing the `rateData.length` in a local variable and referencing it in the requirement and loop on lines 76 and 77 in order to save on the overall cost of gas.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INC-04: Function should be refactored into a modifier

Type	Severity	Location
Implementation	● Informational	contracts/ALP/InterestCalculator.sol L85

Description:

The internal `requireOnlyAdmin` function should be refactored into a modifier.

Recommendation:

Consider refactoring the `requireOnlyAdmin` function into a modifier.

Alleviation:

The recommendation was not applied, with the Fetch.ai team stating "Code style favours functions over modifiers."

INC-05: Contradictory requirement

Type	Severity	Location
Volatile Code	● Medium	contracts/ALP/InterestCalculator.sol L111

Description:

The private `getFracLookup` function in the `InterestCalculator` contract contains a contradictory requirement that the supplied `utilisationRatio` uint256 parameter is greater than or equal to zero, which will always be true regardless of the supplied value due to being unsigned.

Recommendation:

Since unsigned integers cannot be negative, consider refactoring the greater-than-or-equal-to comparion (`>=`) in the requirement into a greater-than comparison (`>`).

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INC-06: Redundant calculation

Type	Severity	Location
Arithmetic	● Informational	contracts/ALP/InterestCalculator.sol L113, L117

Description:

The private `getFracLookup` function in the `InterestCalculator` contract performs a redundant calculation on lines 113 and 117 of subtracting 1 from the `numBins` state variable and multiplying it by the supplied `utilisationRatio` parameter:

```
uint256 _minIndex = (numBins.sub(1)).mul(utilisationRatio).div(10**18);
```

```
uint256 _fracIndex =
(numBins.sub(1)).mul(utilisationRatio).sub(_minIndex.mul(10**18));
```

Recommendation:

Consider storing the result of `numBins.sub(1).mul(utilisationRatio)` in a local `_minUtilisation` variable, then changing the calculation of the local `_minIndex` variable to `_minUtilisation.div(10**18)` and the local `_fracIndex` variable to `_minUtilisation.sub(_minIndex.mul(10**18))` .

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INC-07: Unused named return variables

Type	Severity	Location
Implementation	● Informational	contracts/ALP/InterestCalculator.sol L113-L118

Description:

The private `getFracLookup` function in the `InterestCalculator` contract declares named `minIndex` , `maxIndex` and `fracIndex` return variables, yet declares local `_minIndex` , `_maxIndex` and `_fracIndex` variables and explicitly returns those instead of using the return variables, which is inefficient.

Recommendation:

Consider removing the local `_minIndex` , `_maxIndex` and `_fracIndex` variable declarations and assigning to the named `minIndex` , `maxIndex` and `fracIndex` return variables respectively.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INC-08: Potential integer truncation

Type	Severity	Location
Arithmetic	● Medium	contracts/ALP/InterestCalculator.sol L129-L131

Description:

The public `getBorrowerLnAPRRate` function in the `InterestCalculator` contract performs primitive arithmetic without requiring the values to be valid beforehand, which can result in over/underflow or multiplying/dividing by zero:

```
borrowingRateData[minIndex] +
  (fracIndex * (borrowingRateData[maxIndex] -
borrowingRateData[minIndex])) /
  (10**18)
```

Recommendation:

Since the `SafeMath` library is already imported in the `InterestCalculator` contract, consider using its `add` , `sub` , `mul` and `div` functions in order to prevent over/underflow or multiplying/dividing by zero.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INM-01: Unused named return variables

Type	Severity	Location
Implementation	● Informational	contracts/ALP/InterestManager.sol L84-L90, L92-L99

Description:

The private `generateHashNames` and `generateAllHashNames` functions in the `InterestManager` contract declares a named `hashNames` return variable, yet declares a local `_hashNames` variable and explicitly returns that instead of utilizing the return variable, which is inefficient.

Recommendation:

Consider removing the local `_hashNames` variable declaration and assigning to the named `hashNames` return variable instead.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INM-02: Functions should be re-declared as external

Type	Severity	Location
Gas Optimization	● Informational	contracts/ALP/InterestManager.sol L145, L178

Description:

The public `getSPRRates` and `getLoanSPRRate` functions in the `InterestManager` contract should be re-declared as external.

Recommendation:

Consider re-declaration the public `getSPRRates` and `getLoanSPRRate` functions as external.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INM-03: Unused named return variables

Type	Severity	Location
Implementation	● Informational	contracts/ALP/InterestManager.sol L165-L169

Description:

The public `getSPRRates` function in the `InterestManager` contract declares named `lenderLPSPR` , `borrowerLPSPR` and `borrowerLLSPR` return variables, yet declares local `lenderLPSPR` , `borrowerLPSPR` and `borrowerLLSPR` variables and returns those instead, which is inefficient.

Recommendation:

Consider removing the local `lenderLPSPR` , `borrowerLPSPR` and `borrowerLLSPR` variable declarations and assigning to the named local `lenderLPSPR` , `borrowerLPSPR` and `borrowerLLSPR` return variables respectively.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INM-04: Unused named return variable

Type	Severity	Location
Implementation	● Informational	contracts/ALP/InterestManager.sol L215

Description:

The public `getLoanSPRRate` function in the `InterestManager` contract declares a named `loanSPR` return variable, yet it is never referenced and an explicit return statement is used instead, which is inefficient.

Recommendation:

Consider assigning to the named `loanSPR` return variable instead of using an explicit return statement.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INM-05: Lack of access restriction allows overriding state variables

Type	Severity	Location
Volatile Code	● Critical	contracts/ALP/InterestManager.sol L221-L238

Description:

The public `onRegistryUpdate` function in the `InterestManager` contract does not implement access restriction, which allows anyone to call the function and supply their own `IContractRegistry` value, pausing the system and effectively overriding the `tokenValueStorageContract`, `rateModelLPContract`, `rateModelLLContract`, `lendingPoolContract`, `loanLiquidatorContract`, `utilizationRatioContract` and `spread` state variables with the sender's own supplied values.

Recommendation:

Consider changing the visibility of the base `onRegistryUpdate(IContractRegistry)` function in the `AtomixBase` contract to internal in order to prevent ordinary users from calling it and overriding the state variables of the `InterestManager` contract.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

INM-06: Lack of access restriction allows overriding state variables

Type	Severity	Location
Volatile Code	● Critical	contracts/ALP/InterestManager.sol L244-L260

Description:

The public `onRegistryPostUpdate` function in the `InterestManager` contract does not implement access restriction, which allows anyone to call the function and supply their own `IContractRegistry` value, unpausing the system if the paused state has changed and initializing the `tokenValueStorage` with their own values if it has not already been initialized.

Recommendation:

Consider changing the visibility of the base `onRegistryPostUpdate(IContractRegistry)` function in the `AtomixBase` contract to internal in order to prevent ordinary users from calling it and overriding the state variables of the `InterestManager` contract.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-01: Unnecessary usage of SafeMath functionality

Type	Severity	Location
Implementation	● Informational	contracts/ALP/LendingPool.sol L219

Description:

The public `getUtilizationRatio` function in the `LendingPool` contract performs zero-checks on the local `iSCTotalValue` and `xSCTotalValue` variables before utilizing the `SafeMath.mul` and `SafeMath.div` functions, which is unnecessary and inefficient.

Recommendation:

Since the values are already checked to be valid, consider utilizing primitive multiplication and division operations in order to save on the overall cost of gas.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-02: Functions should be re-declared as external

Type	Severity	Location
Gas Optimization	● Informational	contracts/ALP/LendingPool.sol L226, L323, L353, L492

Description:

The public `getTotalSCDepositValue` , `getXSCValue` , `isDebtOverLimit` , `withdrawAct` functions in the `LendingPool` contract should be re-declared as external.

Recommendation:

Consider re-declaring the public `getTotalSCDepositValue` , `getXSCValue` , `isDebtOverLimit` , `withdrawAct` functions as external.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-03: Potential integer underflow

Type	Severity	Location
Arithmetic	● Minor	contracts/ALP/LendingPool.sol L315

Description:

The public `getAvailableBorrowerLimit` function in the `LendingPool` contract performs a primitive subtraction on the local `totalBorrowingLimit` and `loanValue` variables without checking if their values are valid beforehand, which has the potential for underflow.

Recommendation:

Since the `SafeMath` library is already imported into the `LendingPool` contract, consider utilizing its `sub` function in order to safely protect against integer underflow.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-04: Unused named return variable

Type	Severity	Location
Implementation	● Informational	contracts/ALP/LendingPool.sol L361-L366

Description:

The private `generateHashNames` function in the `LendingPool` contract declares a named `hashNames` return variable, yet declares a local `_hashNames` variable and explicitly returns that instead of utilizing the return variable, which is inefficient.

Recommendation:

Consider removing the local `_hashNames` variable declaration and assigning to the named `hashNames` return variable instead.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-05: Unused result from call to transferFrom

Type	Severity	Location
Volatile Code	● Medium	contracts/ALP/LendingPool.sol L463

Description:

The public `transferSpread` function in the `LendingPool` contract ignores the value returned from the call to the `transferFrom` function.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `SafeERC20` library and utilizing its `safeTransferFrom` function in order to handle ERC-20 implementations which are not fully compliant.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-06: Potential for minor re-entrancy; Out-of-order events

Type	Severity	Location
Volatile Code	● Minor	contracts/ALP/LendingPool.sol L463

Description:

The public `transferSpread` function in the `LendingPool` contract has the potential for re-entrancy due to transfering from the arbitrary `lendingPoolWalletAddress` to the arbitrary `spreadDestinationWallet` address, which can lead to emitting events out of order.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `ReentrancyGuard` contract and utilizing its `nonReentrant` modifier in order to prevent re-entrancy leading to out-of-order events.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-07: Potential for minor re-entrancy; Out-of-order events

Type	Severity	Location
Volatile Code	● Minor	contracts/ALP/LendingPool.sol L478

Description:

The public `depositAct` function in the `LendingPool` contract has the potential for re-entrancy due to transfering from `msg.sender` to the arbitrary `lendingPoolWalletAddress` address, which can lead to emitting events out of order.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `ReentrancyGuard` contract and utilizing its `nonReentrant` modifier in order to prevent re-entrancy leading to out-of-order events.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed, with the Fetch.ai team stating "Used the checks-effects-interactions pattern, Checked that lendingPoolWalletAddress does point to an AtomixWallet and implemented RentrancyGuard."

LPO-08: Potential for minor re-entrancy; Out-of-order events

Type	Severity	Location
Volatile Code	● Minor	contracts/ALP/LendingPool.sol L494

Description:

The public `withdrawAct` function in the `LendingPool` contract has the potential for re-entrancy due to transfering from the arbitrary `lendingPoolWalletAddress` address to `msg.sender`, which can lead to emitting events out of order.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `ReentrancyGuard` contract and utilizing its `nonReentrant` modifier in order to prevent re-entrancy leading to out-of-order events.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-09: Unused result from call to transferFrom

Type	Severity	Location
Volatile Code	● Medium	contracts/ALP/LendingPool.sol L507

Description:

The external `borrow` function in the `LendingPool` contract ignores the value returned from the call to the `transferFrom` function.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `SafeERC20` library and utilizing its `safeTransferFrom` function in order to handle ERC-20 implementations which are not fully compliant.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-10: Potential for minor re-entrancy; Out-of-order events

Type	Severity	Location
Volatile Code	● Minor	contracts/ALP/LendingPool.sol L507

Description:

The external `borrow` function in the `LendingPool` contract has the potential for re-entrancy due to transfering from the arbitrary `lendingPoolWalletAddress` address to `msg.sender` , which can lead to emitting events out of order.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `ReentrancyGuard` contract and utilizing its `nonReentrant` modifier in order to prevent re-entrancy leading to out-of-order events.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-11: Unused result from call to transferFrom

Type	Severity	Location
Volatile Code	● Medium	contracts/ALP/LendingPool.sol L532

Description:

The external `repay` function in the `LendingPool` contract ignores the value returned from the call to the `transferFrom` function.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `SafeERC20` library and utilizing its `safeTransferFrom` function in order to handle ERC-20 implementations which are not fully compliant.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-12: Potential for minor re-entrancy; Out-of-order events

Type	Severity	Location
Volatile Code	● Minor	contracts/ALP/LendingPool.sol L532

Description:

The external `repay` function in the `LendingPool` contract has the potential for re-entrancy due to transfering from the supplied `payee` address parameter to the arbitrary `lendingPoolWalletAddress` address, which can lead to emitting events out of order.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `ReentrancyGuard` contract and utilizing its `nonReentrant` modifier in order to prevent re-entrancy leading to out-of-order events.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-13: Unused result from call to transferFrom

Type	Severity	Location
Volatile Code	● Medium	contracts/ALP/LendingPool.sol L554

Description:

The external `repayAll` function in the `LendingPool` contract ignores the value returned from the call to the `transferFrom` function.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `SafeERC20` library and utilizing its `safeTransferFrom` function in order to handle ERC-20 implementations which are not fully compliant.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-14: Potential for minor re-entrancy; Out-of-order events

Type	Severity	Location
Volatile Code	● Minor	contracts/ALP/LendingPool.sol L554

Description:

The external `repayAll` function in the `LendingPool` contract has the potential for re-entrancy due to transfering from the supplied `payee` address parameter to the arbitrary `lendingPoolWalletAddress` address, which can lead to emitting events out of order.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `ReentrancyGuard` contract and utilizing its `nonReentrant` modifier in order to prevent re-entrancy leading to out-of-order events.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-15: Unused result from call to transferFrom

Type	Severity	Location
Volatile Code	● Medium	contracts/ALP/LendingPool.sol L567

Description:

The external `deposit` function in the `LendingPool` contract ignores the value returned from the call to the `transferFrom` function.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `SafeERC20` library and utilizing its `safeTransferFrom` function in order to handle ERC-20 implementations which are not fully compliant.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-16: Potential for minor re-entrancy; Out-of-order events

Type	Severity	Location
Volatile Code	🔵 Minor	contracts/ALP/LendingPool.sol L567

Description:

The external `deposit` function in the `LendingPool` contract has the potential for re-entrancy due to transfering from the supplied `account` address parameter to the arbitrary `lendingPoolWalletAddress` address, which can lead to emitting events out of order.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `ReentrancyGuard` contract and utilizing its `nonReentrant` modifier in order to prevent re-entrancy leading to out-of-order events.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-17: Unused result from call to transferFrom

Type	Severity	Location
Volatile Code	● Medium	contracts/ALP/LendingPool.sol L592

Description:

The external `redeem` function in the `LendingPool` contract ignores the value returned from the call to the `transferFrom` function.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `SafeERC20` library and utilizing its `safeTransferFrom` function in order to handle ERC-20 implementations which are not fully compliant.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-18: Potential for minor re-entrancy; Out-of-order events

Type	Severity	Location
Volatile Code	● Minor	contracts/ALP/LendingPool.sol L592

Description:

The external `redeem` function in the `LendingPool` contract has the potential for re-entrancy due to transfering from the arbitrary `lendingPoolWalletAddress` address to the supplied `account` address parameter, which can lead to emitting events out of order.

Recommendation:

Since the project imports the `@openzeppelin/contracts` npm module, consider importing the `ReentrancyGuard` contract and utilizing its `nonReentrant` modifier in order to prevent re-entrancy leading to out-of-order events.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LPO-19: Function should be refactored into a modifier

Type	Severity	Location
Implementation	● Informational	contracts/ALP/LendingPool.sol L604, L618, L636, L649, L663, L690

Description:

The private `verifyDepositActAllowed`, `verifyWithdrawActAllowed`, `verifyDepositAllowed`, `verifyRedeemAllowed`, `verifyBorrowAllowed` and `verifyRepayAllowed` functions in the `LendingPool` contract should be refactored into modifiers.

Recommendation:

Consider refactoring the private `verifyDepositActAllowed`, `verifyWithdrawActAllowed`, `verifyDepositAllowed`, `verifyRedeemAllowed`, `verifyBorrowAllowed` and `verifyRepayAllowed` functions in the `LendingPool` contract into modifiers.

Alleviation:

The recommendation was not applied, with the Fetch.ai team stating "Code style favours functions over modifiers."

LPO-20: Lack of access restriction allows overriding state variables

Type	Severity	Location
Volatile Code	● Critical	contracts/ALP/LendingPool.sol L710-L737

Description:

The public `onRegistryUpdate` function in the `LendingPool` contract does not implement access restriction, which allows anyone to call the function and supply their own `IContractRegistry` value, pausing the system and effectively overriding the `actContract` , `stableCoinContract` , `breachMonitorContract` , `lendingPoolStorageContract` , `rateModelLPContract` , `tokenValueStorageContract` , `xSCContract` , `tokenizerContract` , `spread` , `lendingPoolWalletAddress` , `spreadDestinationWallet` , `breachAddress` and `lendingPoolStorageModifierContract` state variables with the sender's own supplied values.

Recommendation:

Consider changing the visibility of the base `onRegistryUpdate(IContractRegistry)` function in the `AtomixBase` contract to internal in order to prevent ordinary users from calling it and overriding the state variables of the `InterestManager` contract.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LSM-01: Lack of access restriction allows overriding state variables

Type	Severity	Location
Volatile Code	● Critical	contracts/ALP/LendingPoolStorageModifier.sol L155-L164

Description:

The public `onRegistryUpdate` function in the `LendingPoolStorageModifier` contract does not implement access restriction, which allows anyone to call the function and supply their own `IContractRegistry` value, pausing the system and effectively overriding the `interestManagerContract` and `lendingPoolStorageContract` state variables with the sender's own supplied values.

Recommendation:

Consider changing the visibility of the base `onRegistryUpdate(IContractRegistry)` function in the `AtomixBase` contract to internal in order to prevent ordinary users from calling it and overriding the state variables of the `InterestManager` contract.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

LSM-02: Function should be refactored into a modifier

Type	Severity	Location
Implementation	● Informational	contracts/ALP/LendingPoolStorageModifier.sol L169

Description:

The private `requireIsPrivileged` function in the `LendingPoolStorageModifier` contract should be refactored as a modifier.

Recommendation:

Consider refactoring the private `requireIsPrivileged` function in the `LendingPoolStorageModifier` contract into a modifier.

Alleviation:

The recommendation was not applied, with the Fetch.ai team stating "Code style favours functions over modifiers."

PSB-01: Function should be re-declared as external

Type	Severity	Location
Gas Optimization	● Informational	contracts/ALP/PoolStorageBase.sol L66

Description:

The public `getLoanDetails` function in the `PoolStorageBase` contract should be re-declared as external.

Recommendation:

Consider re-declaring the public `getLoanDetails` function as external.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

PSB-02: Function should be refactored into a modifier

Type	Severity	Location
Implementation	● Informational	contracts/ALP/PoolStorageBase.sol L194

Description:

The internal `requireLoanExists` function in the `PoolStorageBase` contract should be refactored as a modifier.

Recommendation:

Consider refactoring the internal `requireLoanExists` function in the `PoolStorageBase` contract into a modifier.

Alleviation:

The recommendation was not applied, with the Fetch.ai team stating "Code style favours functions over modifiers."

RLP-01: Unused named return variables

Type	Severity	Location
Implementation	● Informational	contracts/ALP/RateModelLP.sol L32, L34, L50, L52

Description:

The external `calcNewValues` function in the `RateModelLP` contract declares named `finalValueIn` and `finalValueOut` return variables, yet declares local `_finalValueIn` and `_finalValueOut` variables and explicitly returns those instead of using the return variables, which is inefficient.

Recommendation:

Consider removing the local `_finalValueIn` and `_finalValueOut` variable declarations and assigning to the named `finalValueIn` and `finalValueOut` return variables respectively.

Alleviation:

The recommendation was found to be applied as of commit 294675db10f0aeffb7ef442f1a6e320afa3599ed.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Arithmetic

Arithmetic exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.