Lien Protocol: Aggregator

Smart Contracts

Security Assessment

February 24th, 2021

By:
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Sheraz Arshad @ CertiK
sheraz.arshad@certik.org

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 indeed 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	Lien Protocol: Aggregator
Description	Aggregator repository implements functionality to add liquidity for the running term, removing of liquidity along with claiming of rewards, tranching of bonds and liquidation of bonds.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. a2778732be559c55f71f306a9190c5f143506220 2. e5131719f629e93e04c4933498565655abb86241 3. 357ddba673a82848e6d379252610bbac12e24f80

Audit Summary

Delivery Date	February 24th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	February 1st, 2021 - February 24th, 2021

Vulnerability Summary

Total Issues	61 - *50 Resolved, 11 Acknowledged*
🔴 **Total Critical**	0
🟠 **Total Major**	2 - *2 Resolved*
🟡 **Total Medium**	7 - *7 Resolved*
🔵 **Total Minor**	7 - *5 Resolved, 2 Acknowledged*
🟢 **Total Informational**	45 - *36 Resolved, 9 Acknowledged*

Executive Summary

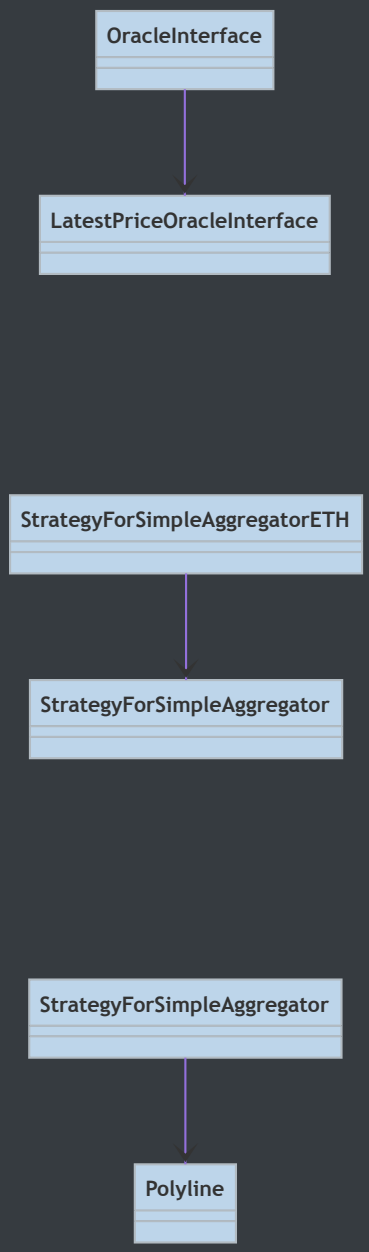
This report represents the results of CertiK's engagement with Lien on their implementation of the Aggregator contracts.

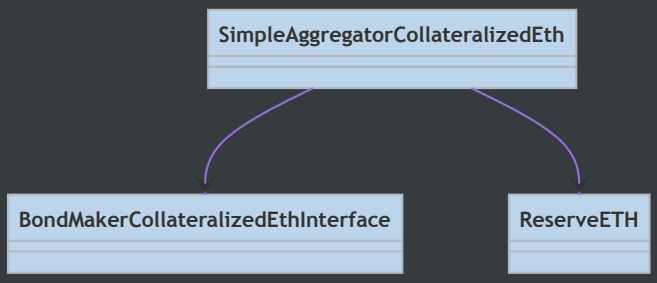
The codebase comprise Aggregator repository which implements functionality to add liquidity for the running term, removing of liquidity along with claiming of rewards, tranching of bonds and liquidation of bonds. Static analysis and manual inspection was performed on the contracts. Our findings include 2 major issues with some medium and minor issues and the rest of the issues are related to code optimizations. Majority of the findings are remediated and the ones that are not considered have a response from the Lien team stated in the Alleviation section of the finding.

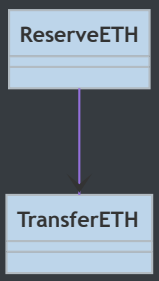
Files In Scope

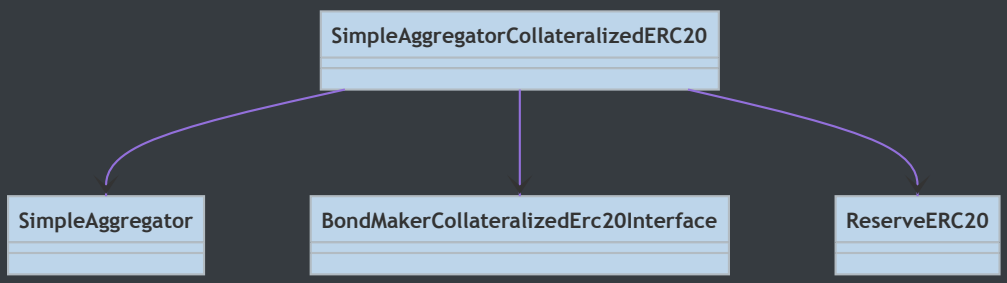
ID	Contract	Location
BRR	BondRegistrator.sol	contracts/SimpleAggregator/BondRegistrator.sol
BRI	BondRegistratorInterface.sol	contracts/Interfaces/BondRegistratorInterface.sol
BPW	BondPricerWithAcceptableMaturity.sol	contracts/SimpleAggregator/BondPricerWithAcceptableMaturity.sol
CGP	CustomGeneralizedPricing.sol	contracts/BondToken_and_GDOTC/bondPricer/CustomGeneralizedPricing.sol
EIE	ExchangeInterface.sol	contracts/Interfaces/ExchangeInterface.sol
LPO	LatestPriceOracleInterface.sol	contracts/Interfaces/LatestPriceOracleInterface.sol
OIE	OracleInterface.sol	contracts/Interfaces/OracleInterface.sol
RET	ReserveETH.sol	contracts/SimpleAggregator/ReserveETH.sol
RER	ReserveERC20.sol	contracts/SimpleAggregator/ReserveERC20.sol
SAR	SimpleAggregator.sol	contracts/SimpleAggregator/SimpleAggregator.sol
SIE	StrategyInterface.sol	contracts/Interfaces/StrategyInterface.sol
SAI	SimpleAggragatorInterface.sol	contracts/Interfaces/SimpleAggragatorInterface.sol
SFS	StrategyForSimpleAggregator.sol	contracts/Strategy/StrategyForSimpleAggregator.sol
SFA	StrategyForSimpleAggregatorETH.sol	contracts/Strategy/StrategyForSimpleAggregatorETH.sol
SAC	SimpleAggregatorCollateralizedEth.sol	contracts/SimpleAggregator/SimpleAggregatorCollateralizedEth.sol
SAE	SimpleAggregatorCollateralizedERC20.sol	contracts/SimpleAggregator/SimpleAggregatorCollateralizedERC20.sol
UVO	UseVolatilityOracle.sol	contracts/Interfaces/UseVolatilityOracle.sol
VOI	VolatilityOracleInterface.sol	contracts/Interfaces/VolatilityOracleInterface.sol

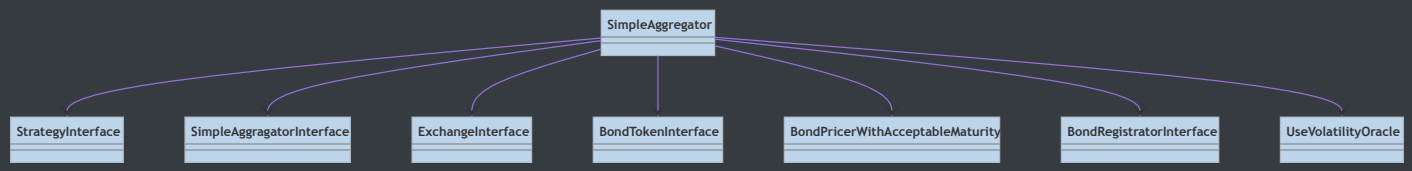
File Dependency Graph

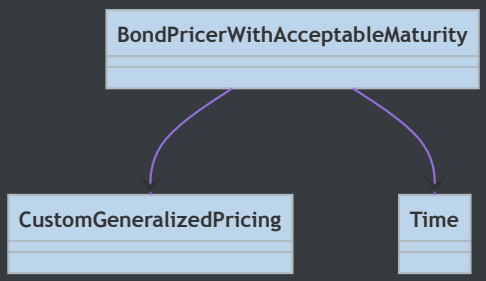


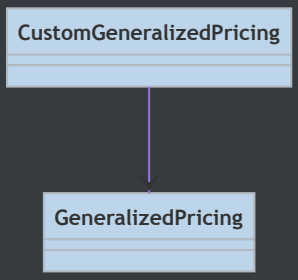

SimpleAggregatorCollateralizedEth
BondMakerCollateralizedEthInterface
ReserveETH

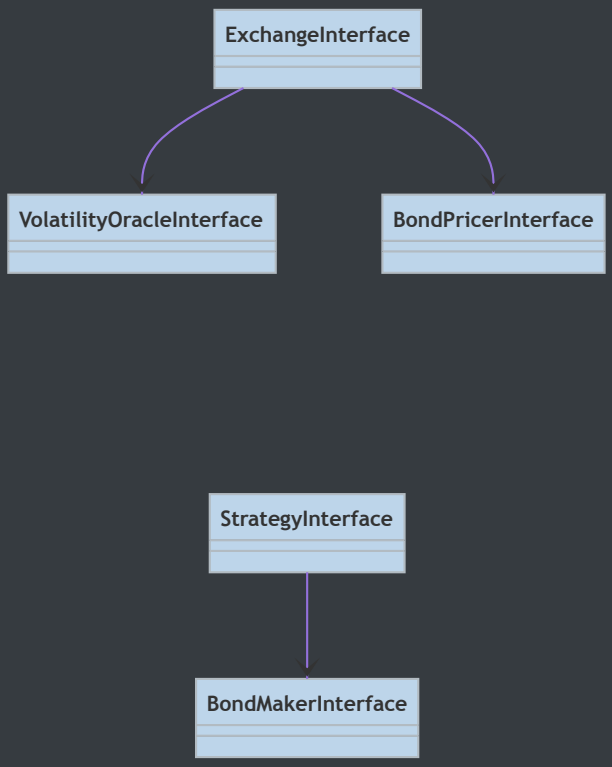

ReserveETH
TransferETH


SimpleAggregatorCollateralizedERC20
SimpleAggregator
BondMakerCollateralizedErc20Interface
ReserveERC20

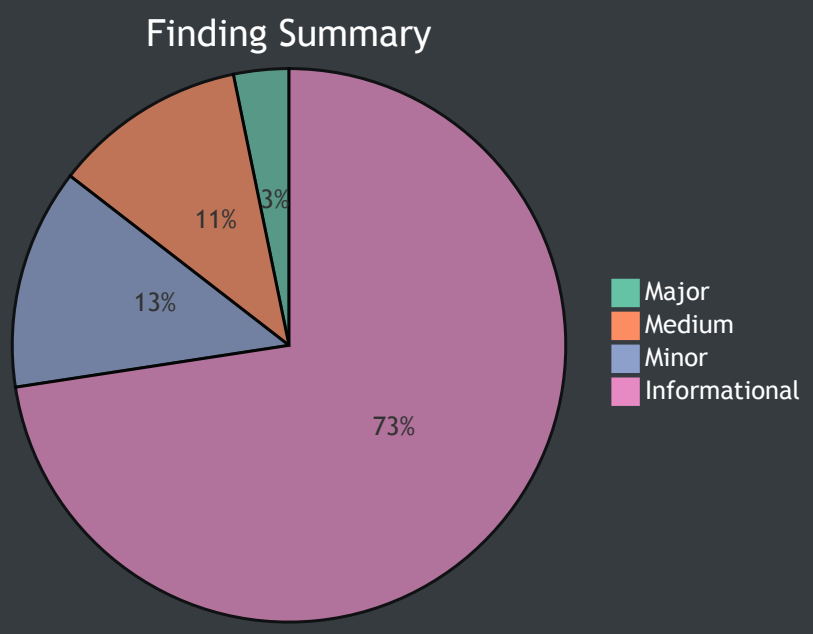

SimpleAggregator
StrategyInterface
SimpleAggragatorInterface
ExchangeInterface
BondTokenInterface
BondPricerWithAcceptableMaturity
BondRegistratorInterface
UseVolatilityOracle


BondPricerWithAcceptableMaturity
CustomGeneralizedPricing
Time


CustomGeneralizedPricing
GeneralizedPricing


ExchangeInterface
VolatilityOracleInterface
BondPricerInterface
StrategyInterface
BondMakerInterface

Findings



ID	Title	Type	Severity	Resolved
SAR-01	Unconventional location of version pragma statement	Language Specific	Informational	✓
SAR-02	Use of relative path for the contracts imported from `node_modules`	Coding Style	Informational	
SAR-03	Mutability Specifiers Missing	Gas Optimization	Informational	✓
SAR-04	Decimals number is hardcoded	Coding Style	Informational	✓
SAR-05	Inefficient storage read	Gas Optimization	Informational	✓

SAR-06	Inefficient storage read	Gas Optimization	Informational	✓
SAR-07	Inefficient storage read	Gas Optimization	Informational	✓
SAR-08	Decimals number is hardcoded	Coding Style	Informational	✓
SAR-09	Inefficient code	Control Flow	Informational	✓
SAR-10	Inefficient code	Control Flow	Informational	✓
SAR-11	Inefficient storage read	Gas Optimization	Informational	✓
SAR-12	Unsafe casting of `uint256` to `uint64`	Mathematical Operations	Minor	✓
SAR-13	Incorrect spellings of the variable name	Coding Style	Informational	✓
SAR-14	Inefficient storage read	Gas Optimization	Informational	✓
SAR-15	Redundant casting to `uint32`	Gas Optimization	Informational	✓
SAR-16	Redundant casting to `uint256`	Gas Optimization	Informational	✓
SAR-17	Inefficient storage lookup	Gas Optimization	Informational	✓
SAR-18	Inefficient storage lookup	Gas Optimization	Informational	✓
SAR-19	`callStrikePrice` is not reversed when the REVERSE_ORACLE is set	Control Flow	Minor	✓
SAR-20	Unsafe subtraction	Mathematical Operations	Medium	✓
SAR-21	Redundant returned variable	Gas Optimization	Informational	✓

SAR-22	Missing return statement from the function's body	Logical Issue	Major	✓
SAR-23	Missing return statement from function's body	Logical Issue	Major	✓
SAR-24	Unused function parameter	Volatile Code	Informational	✓
SAR-25	Incorrect comparison	Logical Issue	Medium	✓
SAR-26	Repeated use of number literals	Coding Style	Informational	✓
SAR-27	Lack of verification for the constructor arguments	Volatile Code	Medium	✓
SAR-28	mappings data can be packed in a struct	Gas Optimization	Informational	
SAR-29	mappings data can be packed in a struct	Gas Optimization	Informational	
SAR-30	Unsafe casting of `uint256` to `uint128`	Mathematical Operations	Minor	✓
SAE-01	Unspecified state variable visibility	Language Specific	Informational	✓
SAE-02	Unspecified state variable visibility	Language Specific	Informational	✓
SAE-03	Unspecified state variable visibility	Language Specific	Informational	✓
SAE-04	Lack of verification for the constructor parameter	Logical Issue	Medium	✓
SAE-05	Inefficient code	Gas Optimization	Informational	✓
SAC-01	Unspecified state variable visibility	Language Specific	Informational	✓
SAC-02	Redundant casting to `address payable`	Gas Optimization	Informational	✓
SAC-	Usage of literal in place of `DECIMAL_GAP`	Coding Style		✓

03			Informational	
SAC-04	Inefficient code	Gas Optimization	Informational	✓
SFS-01	Use of relative path for the contracts imported from `node_modules`	Language Specific	Informational	
SFS-02	Unspecified state variable visibility	Language Specific	Informational	✓
SFS-03	Unspecified state variable visibility	Language Specific	Informational	✓
SFS-04	mappings data can be packed in a struct	Gas Optimization	Informational	
SFS-05	Unsafe arithmetic operations	Mathematical Operations	Minor	✓
SFS-06	Unecessary safe subtraction	Gas Optimization	Informational	✓
SFS-07	Unsafe subtraction	Mathematical Operations	Minor	✓
SFS-08	Unsafe addition	Mathematical Operations	Minor	
SFS-09	Return Variable Utilization	Gas Optimization	Informational	✓
SFS-10	Function Visibility Optimization	Gas Optimization	Informational	✓
SFS-11	Explicitly returning local variable	Gas Optimization	Informational	✓
SFA-01	Unspecified state variable visibility	Language Specific	Informational	✓
SFA-02	Lack of verification of the constructor parameter	Logical Issue	Medium	✓
CGP-01	Naming discrepancy	Inconsistency	Informational	
CGP-02	User-Defined Getters	Gas Optimization	Informational	

BRR-01	Unconventional location of version pragma statement	Language Specific	Informational	✓
BRR-02	Return Variable Utilization	Gas Optimization	Informational	✓
BRR-03	Unsafe casting of `uint256` to `uint64`	Mathematical Operations	Minor	
BPW-01	Use of relative path for the contracts imported from `node_modules`	Coding Style	Informational	
BPW-02	User-Defined Getters	Gas Optimization	Informational	
BPW-03	Missing check for `etherPriceE8` to be greater or equal to zero	Logical Issue	Medium	✓
BPW-04	Missing check for `ethVolatilityE8` to be greater or equal to zero	Logical Issue	Medium	✓

SAR-01: Unconventional location of version pragma statement

Type	Severity	Location
Language Specific	Informational	SimpleAggregator.sol L2-L3

Description:

The aforementioned contract specifies the version pragma statement after the experimental pragma statement which is not the conventional order in Solidity contracts.

Recommendation:

We advise to specify the version pragma statement before the experimental pragma statement to comply with the Solidity convention of pragma statements.

```
pragma solidity 0.7.1;
pragma experimental ABIEncoderV2;
```

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-02: Use of relative path for the contracts imported from `node_modules`

Type	Severity	Location
Coding Style	Informational	SimpleAggregator.sol L11-L14

Description:

The aforementioned lines import contracts from the `node_modules` directory using relative path which is a non-standard way of importing contracts from `node_modules` directory.

Recommendation:

We advise to use the module import path starting at the @ character.

Alleviation:

The recommendation was not taken into account, with the Lien team stating " In our local environment, import path starting with @openzeppelin does not work.".

SAR-03: Mutability Specifiers Missing

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L53-L54

Description:

The linked variables are assigned to only once, either during their contract-level declaration or during the `constructor`'s execution.

Recommendation:

For the former, we advise that the `constant` keyword is introduced in the variable declaration to greatly optimize the gas cost involved in utilizing the variable. For the latter, we advise that the `immutable` mutability specifier is set at the variable's contract-level declaration to greatly optimize the gas cost of utilizing the variables. Please note that the `immutable` keyword only works in Solidity versions `v0.6.5` and up.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-04: Decimals number is hardcoded

Type	Severity	Location
Coding Style	Informational	SimpleAggregator.sol L232, L241

Description:

The aforementioned lines utilize the decimals number as hardcoded value of `8` which can be replaced by `decimals` constant declared as state variable of the contract to increase the legibility of the codebase.

Recommendation:

We advise to utilize the `decimals` constant instead of hardcoding the decimals number on the aforementioned lines.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-05: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L223, L227, L245

Description:

The aforementioned lines perform repeated storage reads for `shareData[currentTerm].totalShare` which can be optimized to consume less gas by storing the result in a local variable and utilizing it.

Recommendation:

We advise to store `shareData[currentTerm].totalShare` in a local variable and utilize it instead of repeatedly reading from storage which consumes significant gas for each storage read oepration.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-06: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L220

Description:

The function `renewMaturity` on the aforementioned line reads `currentTerm` from storage repeatedly in its body. It is inefficient as reading from storage costs around `800` gas each time the read operation is performed. The cost of function execution can be significantly reduced by using a local variable to store the `currentTerm` and then utilizing it instead of reading from storage each time.

Recommendation:

We advise to utilize a local variable to store `currentTerm` and then use it in the aforementioned function instead of reading from storage every time it is needed.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-07: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L253, L267

Description:

The aforementioned lines perform the same storage read twice.

Recommendation:

We advise to limit to the storage read to only one and store the value in a local variable as reading from local variable is significantly cheaper than reading from storage.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-08: Decimals number is hardcoded

Type	Severity	Location
Coding Style	Informational	SimpleAggregator.sol L290

Description:

The aforementioned line has decimals number hardcoded which can be replaced by the state constant `decimals` to increase the legibility of the codebase. Additionally, the amount specifying the totalSupply of `LIEN` does not separate the decimlas from the token amount which can be confusing to read without knowing the decimals number.

Recommendation:

We advise to to make use of decimals constant on the aforementioned line to increase the legibility of the codebase.

```
    require(
        rewardRate > 10 ** decimals && rewardRate < 1000000 * 10 ** decimals,
        "Out of valid reward rate range"
    );
```

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-09: Inefficient code

Type	Severity	Location
Control Flow	Informational	SimpleAggregator.sol L335-L347

Description:

The code on the aforementioned lines is inefficient and redundant which can be optimized by introducing a single `if` statement which sets the `currentTerm` when the `receivedCollaterals` for an address is zero.

Recommendation:

We advise to optimize the code on the aforementioned lines by replacing `if-else` block by a single `if` block which sets the `currentTerm` when the `receievedCollaterals` for the user is zero. When the `receievedCollateral` for the user is zero, it implies that the current deposit is for a newer term and it can be updated as such. The `currentTerm` is accessed in the aforementioned function repeatedly from storage which can be stored in a local variable to save gas cost. Additionally, the return of `false` at the end of the function can be removed as it is not reachable becuause there are only two possibilities of either the older collaterals are moved setting `value` to zero or collateral is from the `currentTerm`.

```
uint256 _currentTerm = currentTerm;
if (receivedCollaterals[msg.sender].value == 0) {
   receivedCollaterals[msg.sender].term = uint128(_currentTerm);
}
receivedCollaterals[msg.sender].value += uint128(amount);
totalReceivedCollateral[_currentTerm] += amount.toUint128();
emit SetAddLiquidity(msg.sender, _currentTerm, amount);
return true;
```

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-10: Inefficient code

Type	Severity	Location
Control Flow	Informational	SimpleAggregator.sol L363-L393

Description:

The code on the aforementioned line is inefficient such that the `if-else` block can be replaced by a single `if` block which sets the `currentTerm` when the `unremovedTokens` for the user is zero.

Recommendation:

We advise to replaced the `if-else` block on the aforementioned by a single `if` block which sets the `currentTerm` when `unremoved` tokens for the user is zero. The `currentTerm` can be stored in a local variable to avoid expensive repeated storage reads. Additionally, the code on `L387-L393` is unreachable because there only two possibilities of either the `value` is zero when tokens are removed from the older term or when the tokens are being removed from the `currentTerm`.

```
uint256 _currentTerm = currentTerm;
if (unremovedTokens[msg.sender].value == 0) {
   unremovedTokens[msg.sender].term = uint128(_currentTerm);
}
unremovedTokens[msg.sender].value += amount;
totalUnremovedTokens[_currentTerm] += amount;
_updateBalanceDataForLiquidityMove(
        msg.sender,
        unsentToken,
        amount,
        addLiquidityTerm
);
emit SetRemoveLiquidity(msg.sender, _currentTerm, amount);
return true;
```

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-11: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L396

Description:

The function `_settleTokens` on the aforementioned line performs several redundant storage reads which results increase gas cost associated with function execution.

Recommendation:

We advise to make to use of local variables to store `currentTerm`, `` `unremovedTokens[msg.sender].value ``, `unremovedTokens[msg.sender].term`, `receivedCollaterals[msg.sender].value` and `receivedCollaterals[msg.sender].term` so multiple reads from the storage could avoided as they result in increased gas cost.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-12: Unsafe casting of `uint256` to `uint64`

Type	Severity	Location
Mathematical Operations	Minor	SimpleAggregator.sol L923

Description:

The aforementioned line performs unsafe casting of `uint256` to `uint64` resulting in truncated value if it overflows `uint64`.

Recommendation:

We advise to utilize the `toUint64` function from SafeCast library to perform the safe casting of `uint256` to `uint64`.

Alleviation:

The relevant code part is removed rendering this exhibit ineffectual.

SAR-13: Incorrect spellings of the variable name

Type	Severity	Location
Coding Style	Informational	SimpleAggregator.sol L473, L474, L477, L504

Description:

The aforementioned lines specify the incorrect spellings for the word `previous` being used as variable name.

Recommendation:

We advise to rectify the spellings of the aforementioned word to increase the legibility of the codebase.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-14: Inefficient storage read

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L489

Description:

The function `liquidateBonds` on the aforementioned lines performs repeated reads of `currentTerm` from storage and it is inefficient as each read operations costs 800 gas.

Recommendation:

We advise to store the `currentTerm` in a local variable and utilized to reduce gas cost.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-15: Redundant casting to `uint32`

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L548-L550

Description:

The aforementioned line performs redundant casting to `uint32`.

Recommendation:

We advise to remove the redundant casting to `uint256` to avoid gas cost associated with it.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-16: Redundant casting to `uint256`

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L586

Description:

The aforementioned line performs redundant casting to `uint256`.

Recommendation:

We advise to remove the redundant casting to `uint256` to avoid gas cost associated with it.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-17: Inefficient storage lookup

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L706, L713, L714, L715

Description:

The aforementioned lines perform inefficient storage lookup which can optimized be copying the `TermInfo` struct from storage to memory and then utilizing it. It will result in reduced gas cost.

Recommendation:

We advise to utilize the struct for storing `TermInfo` value from storage to reduce the gas cost associated with lookup and storage read.

```
uint256 _currentTerm = currentTerm;
TermInfo memory info = termInfo[_currentTerm];

callStrikePrice = _adjustPrice(
    info.strikePrice,
    callStrikePrice
);

bondGroupID = BOND_REGISTRATOR.registerBondGroup(
    BONDMAKER,
    callStrikePrice,
    info.strikePrice,
    info.maturity,
    info.SBTId
);
```

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-18: Inefficient storage lookup

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L666-L691

Description:

The aforementioned lines perform inefficient storage look up which can optimized by declaring a local mapping pointing to storage having roundPrice as key and bondGroupId as value.

Recommendation:

We advise to make use of local mapping pointing to storage to reduce lookup gas cost.

```
mapping(uint256 => uint256) storage priceToGroupBondId =
strikePriceToBondGroup[currentTerm];
if (priceToGroupBondId[roundedPrice] != 0) {
   return priceToGroupBondId[roundedPrice];
}
// Suitable bond range is in between current price +- 2 * priceUnit
for (uint256 i = 1; i <= 2; i++) {
   if (
         priceToGroupBondId[
                   roundedPrice - priceUnit * i
               ] != 0
        ) {
               return
                   priceToGroupBondId[
                       roundedPrice - priceUnit * i
                   ];
        }

        if (
               priceToGroupBondId[
                   roundedPrice + priceUnit * i
               ] != 0
```

```
        ) {
                return
                    priceToGroupBondId[
                        roundedPrice + priceUnit * i
                    ];
        }
    }
```

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-19: `callStrikePrice` is not reversed when the REVERSE_ORACLE is set

Type	Severity	Location
Control Flow	Minor	SimpleAggregator.sol L718-L723

Description:

The aforementioned lines execute when `REVERSE_ORACLE` is set but the call `_addBondGroup` on `L719` passes non-reversed value as its second argument while it should pass a reversed value because it is called when the `REVERSE_ORACLE` is set.

Recommendation:

We advise to double check that if it is an intended behaviour. To get the reversed price, the function `calcCallStrikePrice` on `STRATEGY` contract can be called.

```
if (REVERSE_ORACLE) {
    _addBondGroup(
      bondGroupID,
      STRATEGY.calcCallStrikePrice(
         currentPriceE8,
         priceUnit,
         REVERSE_ORACLE
      );
    );
}
```

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`, with Lien team stating "make REVERSE_ORACLE false as it can be used in _getSuitableBondGroup()".

SAR-20: Unsafe subtraction

Type	Severity	Location
Mathematical Operations	Medium	SimpleAggregator.sol L795

Description:

The aforementioned line performs unsafe subtraction which can result in underflow of the resultant value.

Recommendation:

We advise to utilize the `sub` function from `SafeMath` library to perform subtraction which will revert in the event when `sbtStrikePrice` is greater than `callStrikePrice`.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-21: Redundant returned variable

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L893

Description:

The function `_calcNextReward` on the aforementioned line performs operations on its parameter `balanceData` and returns at the end of function's execution. As the parameter is of reference type, any changes made to the parameter are reflected in the calling function rendering the returning of this variable redundant.

Recommendation:

We advise to remove the return value from the signature of the function as changes made to `BalanceData` are reflected in the calling function.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-22: Missing return statement from the function's body

Type	Severity	Location
Logical Issue	Major	SimpleAggregator.sol L945

Description:

The function `approve` on the aforementioned line is missing the return statement resulting in the function returning default `false` value for every call.

Recommendation:

We advise to add a return statement at the end of function returning literal `true`.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-23: Missing return statement from function's body

Type	Severity	Location
Logical Issue	Major	SimpleAggregator.sol L985

Description:

The `transferFrom` function is missing return statement in its body resulting in the returned value being evaluated to default `false` for every function call.

Recommendation:

We advise to place the `return` before call to the function `_transferToken`, so the result of this call is returned by the function.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-24: Unused function parameter

Type	Severity	Location
Volatile Code	Informational	SimpleAggregator.sol L1162, L1170

Description:

The function parameter `hasReservation` does not affect the functionality of the function and can be removed.

Recommendation:

We advise to remove the unused function parameter `hasReservation` from the body of the function to increase the legibility of the codebase.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-25: Incorrect comparison

Type	Severity	Location
Logical Issue	Medium	SimpleAggregator.sol L1171

Description:

The comparison on the aforementioned line compares `currentTerm` with the term of user's last removal of tokens, which is incorrect and should be replaced with the comparison where `currentTerm` is compared with the term of user's last receiving of collateral.

Recommendation:

We advise to rectify the comparison on the aforementioned line to be compared with the term of user's last receiving of collateral instead of the term of last removal of tokens.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-26: Repeated use of number literals

Type	Severity	Location
Coding Style	Informational	SimpleAggregator.sol L1194, L147

Description:

The aforementioned lines specify the cool time in literal numbers, which can be placed in a state constant and utilized to increase the legibility of the codebase.

Recommendation:

We advise to store the literal numbers representing cool time, on the aforementioned in a state constant and then utilized.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241` .

SAR-27: Lack of verification for the constructor arguments

Type	Severity	Location
Volatile Code	Medium	SimpleAggregator.sol L188

Description:

The constructor on the aforementioned lines does not verify the arguments it receives against zero value before assigning them to `immutable` state variables which cannot be changed once the contract is deployed. Additionally, the parameter `_firstRewardRate` is not validated against its allowed range of values.

Recommendation:

We advise to check the constructor args against their zero values and parameter `_firstRewardRate` should be checked to be satisfying the condition `rewardRate > 10**8 && rewardRate < 10**14`.

Alleviation:

alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAR-28: mappings data can be packed in a struct

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L65, L67, L68, L73, L74, L91, L92

Description:

The mappings on the aforementioned lines have key of type `uint256` representing current term. These mappings can be combined into a single mapping having `uint256` as key type and the value type will be a struct having properties from all aforementioned mappings. This will reduce the lookup gas cost when reading data from these mappings.

Recommendation:

We advise to replace the aforementioned mappings with a single mapping by utilizing a struct for the value types across all the aforementioned mappings.

Alleviation:

The recommendation was not considered, with the Lien team stating "The modification requires a lot of efforts comparing to the benefit from gas optimization".

SAR-29: mappings data can be packed in a struct

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregator.sol L82-L85

Description:

The mappings on the aforementioned lines have key of type address representing a user's `address`. These mappings can be combined into a single mapping having `address` as key type and the value type will be a struct having properties from all of the aforementioned mappings. This will reduce the lookup gas cost when reading data from these mappings.

Recommendation:

We advise to replace the aforementioned mappings with a single mapping by utilizing a struct for the value types across all the aforementioned mappings.

Alleviation:

The recommendation was not considered, with the Lien team stating "The modification requires a lot of efforts comparing to the benefit from gas optimization".

SAR-30: Unsafe casting of `uint256` to `uint128`

Type	Severity	Location
Mathematical Operations	Minor	SimpleAggregator.sol L336, L337, L342

Description:

The aforementioned lines perform unsafe casting of `uint256` values `uint128` resulting in truncated value if it overflows `uint128`.

Recommendation:

We advise to utilize the `toUint128` function from `SafeCast` library from Openzeppelin to perform safe cast of `uint256` value to `uint128`.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAE-01: Unspecified state variable visibility

Type	Severity	Location
Language Specific	Informational	SimpleAggregatorCollateralizedERC20.sol L12

Description:

The `collateralToken` state variable in the `SimpleAggregatorCollateralizedERC20` contract should have its visibility specified.

Recommendation:

Consider specifying the visibility of the `collateralToken` state variable in the `SimpleAggregatorCollateralizedERC20` contract as internal or private.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAE-02: Unspecified state variable visibility

Type	Severity	Location
Language Specific	Informational	SimpleAggregatorCollateralizedERC20.sol L13

Description:

The `decimalGap` state variable in the `SimpleAggregatorCollateralizedERC20` contract should have its visibility specified.

Recommendation:

Consider specifying the visibility of the `decimalGap` state variable in the `SimpleAggregatorCollateralizedERC20` contract as internal or private.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAE-03: Unspecified state variable visibility

Type	Severity	Location
Language Specific	Informational	SimpleAggregatorCollateralizedERC20.sol L14

Description:

The `reserveERC20` state variable in the `SimpleAggregatorCollateralizedERC20` contract should have its visibility specified.

Recommendation:

Consider specifying the visibility of the `reserveERC20` state variable in the `SimpleAggregatorCollateralizedERC20` contract as internal or private.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAE-04: Lack of verification for the constructor parameter

Type	Severity	Location
Logical Issue	Medium	SimpleAggregatorCollateralizedERC20.sol L23

Description:

The constructor parameter on the aforementioned is not validated against zero and assigned to `collateralToken` state variable which is then not changeable.

Recommendation:

We recommend to add a check asserting that the parameter on the aforementioned line is not zero.

```
    require(
       address(_collateralAddress) != address(0),
       "collateralAddress cannot be zero"
    );
```

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAE-05: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregatorCollateralizedERC20.sol L99, L105, L126, L133, L141

Description:

The `currentFeeBase` storage variable is updated on `L99` and then subsequently read on `L105, L126, L133 and L141` . As reading from storage costs 800 gas each time it is performed, it is advisable to store the returned value from function call on `L99` in a local variable and then utilize it to update storage variable and as well as in the rest of function's body.

Recommendation:

We advise to store the returned value from function call on `L99` in a local variable and then utilize it to update storage and place the local variable where the strorage reads are currently performed through `currentFeeBase` .

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241` .

SAC-01: Unspecified state variable visibility

Type	Severity	Location
Language Specific	Informational	SimpleAggregatorCollateralizedEth.sol L12

Description:

The `reserveEth` state variable in the `SimpleAggregatorCollateralizedEth` contract should have its visibility specified.

Recommendation:

Consider specifying the visibility of the `reserveEth` state variable in the `SimpleAggregatorCollateralizedEth` contract as internal or private.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAC-02: Redundant casting to `address payable`

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregatorCollateralizedEth.sol L123

Description:

The aforementioned line performs redundant casting using `payable()` as casting of contract type variable to `address` already results in `address payable` if the contract contains `receive` or `payable fallback` function.

Recommendation:

We advise to remove the redundant `payable` casting on the aforementioned line to save gas cost associated with it.

```
_transferETH(address(reserveEth), amount);
```

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAC-03: Usage of literal in place of `DECIMAL_GAP`

Type	Severity	Location
Coding Style	Informational	SimpleAggregatorCollateralizedEth.sol L132

Description:

The aforementioned line utilizes integer literal in place of `DECIMAL_GAP`.

Recommendation:

We advise to make to use of `DECIMAL_GAP` in place integer literal to increase the legibility of the codebase.

```
bm.issueNewBonds{value: amount.mul(10**DECIMAL_GAP).mul(1002).div(1000)}(
            bondgroupID
);
```

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SAC-04: Inefficient code

Type	Severity	Location
Gas Optimization	Informational	SimpleAggregatorCollateralizedEth.sol L65, L72, L85, L88, L95

Description:

The return value from function call on `L65` is assigned to the storage variable `currentFeeBase` and then it is subsequently read several times in the rest of the function.

Recommendation:

We advise to store the returned value from function call on `L65` in a local variable and then subsequently use the local variable in the rest of the function so increased gas cost associated with reading from storage could be avoided.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFS-01: Use of relative path for the contracts imported from `node_modules`

Type	Severity	Location
Language Specific	Informational	StrategyForSimpleAggregator.sol L9-L10

Description:

The aforementioned lines import contracts from the `node_modules` directory using relative path which is a non-standard way of importing contracts from `node_modules` directory.

Recommendation:

We advise to use the module import path starting at the @ character.

Alleviation:

The recommendation was not considered, with Lien team stating "In our local environment, import path starting with @openzeppelin does not work".

SFS-02: Unspecified state variable visibility

Type	Severity	Location
Language Specific	Informational	StrategyForSimpleAggregator.sol L23

Description:

The `TERM_INTERVAL` state variable in the `StrategyForSimpleAggregator` contract should have its visibility specified.

Recommendation:

Consider specifying the visibility of the `TERM_INTERVAL` state variable in the `StrategyForSimpleAggregator` contract as internal or private.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFS-03: Unspecified state variable visibility

Type	Severity	Location
Language Specific	Informational	StrategyForSimpleAggregator.sol L24

Description:

The `TERM_CORRECTION_FACTOR` state variable in the `StrategyForSimpleAggregator` contract should have its visibility specified.

Recommendation:

Consider specifying the visibility of the `TERM_CORRECTION_FACTOR` state variable in the `StrategyForSimpleAggregator` contract as internal or private.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFS-04: mappings data can be packed in a struct

Type	Severity	Location
Gas Optimization	Informational	StrategyForSimpleAggregator.sol L21-L22

Description:

The mappings on the aforementioned lines have key of type `bytes32` representing aggregator id. These mappings can be combined into a single mapping having `bytes32` as key type and the value type will be a struct having properties from all aforementioned mappings. This will reduce the lookup gas cost when reading data from these mappings.

Recommendation:

We advise to replace the aforementioned mappings with a single mapping by utilizing a struct for the value types across all the aforementioned mappings.

Alleviation:

Then recommendation was not considered, with Lien team stating "The modification requires a lot of efforts comparing to the benefit from gas optimization".

SFS-05: Unsafe arithmetic operations

Type	Severity	Location
Mathematical Operations	Minor	StrategyForSimpleAggregator.sol L84-L85

Description:

The aforementioned lines perform unsafe arithmetic operations that can result in underflow or overflow integer values.

Recommendation:

We advise to make use of library SafeMath's functions `add`, `sub` and `mul` to perform arithmetic operations on the aforementioned lines.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFS-06: Unecessary safe subtraction

Type	Severity	Location
Gas Optimization	Informational	StrategyForSimpleAggregator.sol L112

Description:

The aforementioned line performs unnecessary safe subtraction as the expression can never underflow with `balance` always being less than `baseAmount` becuase of the `if` condition on `L111`.

Recommendation:

We advise to perform the simple subtraction instead of safe subtraction to avoid extra gas cost associated with it.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFS-07: Unsafe subtraction

Type	Severity	Location
Mathematical Operations	Minor	StrategyForSimpleAggregator.sol L169

Description:

The aforementioned line performs unsafe subtraction which can result in underflow of the value resulting from the expression.

Recommendation:

We advise to add a check asserting that `currentFeeBase` is greater than or equal to `downwardDifference` so the function reverts in case of underflow.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFS-08: Unsafe addition

Type	Severity	Location
Mathematical Operations	Minor	StrategyForSimpleAggregator.sol L173

Description:

The aforementioned line performs unsafe addition which can result in overflow integer value resulting from expression.

Recommendation:

We advise to add a check asserting that the sum of `currentFeeBase` and `upwardDifference` is greater than `currentFeeBase` so that the function call reverts in case of overflow.

```
require(
   currentFeeBase + upwardDifference >= currentFeeBase,
   "overflow occured"
);
```

Alleviation:

The recommendation was not considered, with Lien team stating "Unnecessary modification: current fee base is under 1000".

SFS-09: Return Variable Utilization

Type	Severity	Location
Gas Optimization	Informational	StrategyForSimpleAggregator.sol L159

Description:

The linked function declarations contain explicitly named `return` variables that are not utilized within the function's code block.

Recommendation:

We advise that the linked variables are either utilized or omitted from the declaration.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFS-10: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	StrategyForSimpleAggregator.sol L195

Description:

The linked function is declared as `public`, contains array function arguments and is not invoked in any of the contract's contained within the project's scope.

Recommendation:

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFS-11: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	Informational	StrategyForSimpleAggregator.sol L248

Description:

The function on the aforementioned line explicitly returns a local variable which increases the overall cost of gas.

Recommendation:

Since named return variables can be declared in the signature of a function, consider refactoring to remove the local variable declaration and explicit return statement in order to reduce the overall cost of gas.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFA-01: Unspecified state variable visibility

Type	Severity	Location
Language Specific	Informational	StrategyForSimpleAggregatorETH.sol L7

Description:

The `exchange` state variable in the `StrategyForSimpleAggregatorETH` contract should have its visibility specified.

Recommendation:

Consider specifying the visibility of the `exchange` state variable in the `StrategyForSimpleAggregatorETH` contract as internal or private.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

SFA-02: Lack of verification of the constructor parameter

Type	Severity	Location
Logical Issue	Medium	StrategyForSimpleAggregatorETH.sol L9

Description:

The constructor parameter of `_exchange` is not checked against zero value before it is assigned to the state variable `exchange`. As the state variable `exchange` is immutable, so it cannot be reassigned if it is once assigned with a zero value.

Recommendation:

We recommend to add a check asserting that the `_exhange` is not zero.

```
require(
   address(_exchange) != address(0),
   "_exchange cannot be zero"
);
```

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

CGP-01: Naming discrepancy

Type	Severity	Location
Inconsistency	Informational	CustomGeneralizedPricing.sol L54, L78

Description:

The error messages on the aforementioned lines incorrectly refer to `stable bond` as `solid bond`.

Recommendation:

We advise to rectify the error messages on the aforementioned lines to correctly refer to stable bonds.

Alleviation:

The recommendation was not considered and the Lien team did not provide a response on it suggesting that the finding is ineffectual.

CGP-02: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	CustomGeneralizedPricing.sol L10

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (_) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The recommentation was not considered.

BRR-01: Unconventional location of version pragma statement

Type	Severity	Location
Language Specific	Informational	BondRegistrator.sol L2-L3

Description:

The aforementioned contract specifies the version pragma statement after the experimental pragma statement which is not the conventional order in Solidity contracts.

Recommendation:

We advise to specify the version pragma statement before the experimental pragma statement to comply with the Solidity convention of pragma statements.

```
pragma solidity 0.7.1;
pragma experimental ABIEncoderV2;
```

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

BRR-02: Return Variable Utilization

Type	Severity	Location
Gas Optimization	Informational	BondRegistrator.sol L9

Description:

The linked function declarations contain explicitly named `return` variables that are not utilized within the function's code block.

Recommendation:

We advise that the linked variables are either utilized or omitted from the declaration.

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

BRR-03: Unsafe casting of `uint256` to `uint64`

Type	Severity	Location
Mathematical Operations	Minor	BondRegistrator.sol L66, L68, L72, L74, L76, L77, L88, L92, L94, L105, L110

Description:

The aforementioned lines perform unsafe casting of `uint256` values `uint64` resulting in truncated value if it overflows `uint64`.

Recommendation:

We advise to utilize the `toUint64` function from `SafeCast` library from Openzeppelin to perform safe cast of `uint256` value to `uint64`.

Alleviation:

The recommendation was not considered, with Lien team stating "These points are checked in bondmaker contract.".

BPW-01: Use of relative path for the contracts imported from `node_modules`

Type	Severity	Location
Coding Style	Informational	BondPricerWithAcceptableMaturity.sol L6

Description:

The aforementioned line imports contract from the `node_modules` directory using relative path which is a non-standard way of importing contracts from `node_modules` directory.

Recommendation:

We advise to use the module import path starting at the @ character.

Alleviation:

The recommendation was not considered, with Lien team stating "In our local environment, import path starting with @openzeppelin does not work".

BPW-02: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	BondPricerWithAcceptableMaturity.sol L16

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (_) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The recommendation was not considered.

BPW-03: Missing check for `etherPriceE8` to be greater or equal to zero

Type	Severity	Location
Logical Issue	Medium	BondPricerWithAcceptableMaturity.sol L115

Description:

The comparison on the aforementioned line should be extended with another comparison for `etherPriceE8` not to be less than zero.

Recommendation:

We advise to add check for `etherPriceE8` to be not less than zero.

```
require(
    etherPriceE8 >= 0 && etherPriceE8 < 100000 * 10**8,
    "ETH price should be between $0 and $100000"
);
```

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

BPW-04: Missing check for `ethVolatilityE8` to be greater or equal to zero

Type	Severity	Location
Logical Issue	Medium	BondPricerWithAcceptableMaturity.sol L119

Description:

The comparison on the aforementioned line should be extended with another comparison for `ethVolatilityE8` not to be less than zero.

Recommendation:

We advise to add check for `ethVolatilityE8` to be not less than zero.

```
    require(
        ethVolatilityE8 >= 0 && ethVolatilityE8 < 10 * 10**8,
        "ETH volatility should be between 0% and 1000%"
    );
```

Alleviation:

Alleviations were applied as of commit hash `e5131719f629e93e04c4933498565655abb86241`.

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.

Mathematical Operations

Mathematical Operation 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 invokeable 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.