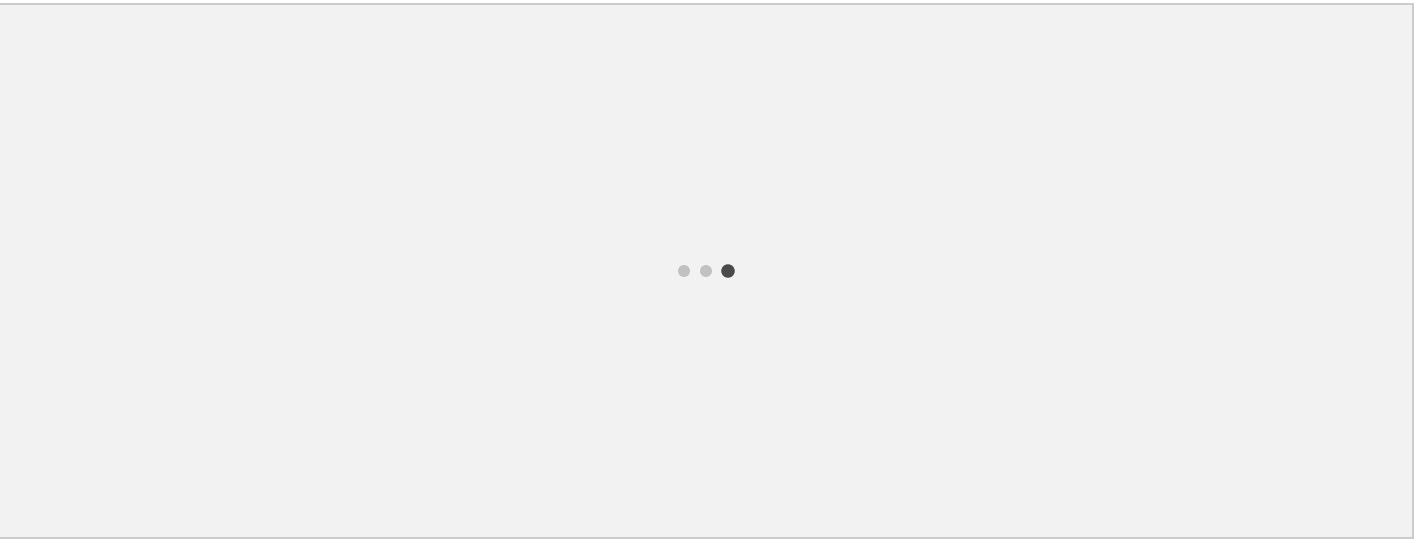

Contents

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Lien (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles

such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for **Lien** to discover issues and vulnerabilities in the source code of their stablecoin and call option **Smart Contracts** as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

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

Testing Summary

SECURITY LEVEL

Smart Contract Audit

This report has been prepared as a product of the Smart Contract Audit request by Lien.

This audit was conducted to discover issues and vulnerabilities in the source code of Lien Protocol Contracts.

TYPE	Smart Contracts
SOURCE CODE	https://github.com/LienFinance/idol https://github.com/LienFinance/fairswap https://github.com/LienFinance/lien-token https://github.com/LienFinance/oracle
PLATFORM	EVM
LANGUAGE	Solidity
REQUEST DATE	June 25, 2020
FINAL DELIVERY DATE	September 04, 2020
METHODS	A comprehensive examination has been performed using Dynamic Analysis, Static Analysis, and Manual Review.

Review Notes

Introduction

CertiK team was contracted by the Lien team to audit the design and implementations of their smart contracts that are meant to be utilized in correlation to form the iDOL stablecoin as well as provide a way to users to hold Vickrey auctions for Ether.

The audited source code links are:

- iDOL: https://github.com/LienFinance/idol/tree/a02a5e3b86097babaf8efca20df38d5ae33fb120
- Fairswap: https://github.com/LienFinance/fairswap/tree/dfa5f4b9f221afa809aa8af5c41775021a4c4ab9
- Lien Token: https://github.com/LienFinance/lien-token/tree/89f77f593d4529d26246339222ad90c70b11e7ca
- Oracle: https://github.com/LienFinance/oracle/tree/696d01c3bc29e2ce416a5ba759181ebc998f1038

The goal of this audit was to review the Solidity implementation for its business model, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

The findings of the initial audit have been conveyed to the team behind the contract implementations and the source code is expected to be re-evaluated before another round of auditing has been carried out.

A second round of auditing was carried out in the following commit hashes that includes remediations as well as minor changes based on feedback from their mainnet launch. These commit hashes are as follows:

- iDOL: https://github.com/LienFinance/idol/tree/d44b26ba9f7fd42bdebeb7f85799d287f1ec927f
- Lien Token: https://github.com/LienFinance/lien-token/tree/aafe4c5998352a5aca8a27bdbb028ace9a1acb9a
- Oracle: https://github.com/LienFinance/oracle/tree/e90c9cad3e9d40a2bb994894dd361af16f1415b9

Documentation

The sources of truth regarding the operation of the contracts in scope were comprehensive and **greatly aided our efforts to audit the project as well as generally increase the legibility of the codebase**. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behaviour, we consulted with the Sandbox team or reported an issue.

Summary

We began the audit by inspecting the source code of the Lien Token, Oracle and Fairswap repositories before moving forward with the core iDOL audit as we noticed the iDOL repository to heavily depend on the aforementioned modules.

During the second phase of the two week audit we analyzed the code of the core protocol within iDOL as well as delved into greater depth on the Fairswap repository to identify any potential vulnerabilities, misalignments with the specification and unaccounted for functionalities / behaviors.

Findings are split into four categories, specifically `INFORMATIONAL`, `MINOR`, `MEDIUM` and `MAJOR`. Of those four, the `INFORMATIONAL` category contains exhibits that can be safely ignored as they relate to code optimizations, misspellings and cosmetic changes based on the Solidity style guide that under most cases do not result in a change in the generated bytecode.

Recommendations

We advise that all “MINOR” and up Exhibits are dealt with and that all “INFORMATIONAL” Exhibits are assessed and assimilated into the codebase wherever sensible and feasible. Overall, the Lien team has demonstrated an in-depth understanding of the mathematical formulas involved in the solution they aspire to launch and showcased healthy code ethics within each project’s codebase.

Conclusion

We went through an iterative approach with the Lien team to remediate most of the optimization findings we pointed out as well as all the vulnerabilities and mathematical discrepancies we were able to identify within their codebase.

We maintained a direct real-time communication channel with the Lien team to ease the process of evaluating the remediations as well as go through multiple revisions of a change when necessary. The Lien team was highly responsive and provided code changes for the remediations in a very timely manner.

While the source code we audited was not the linked ones of the report but mirrored private repositories that served as development instances, we verified each file's contents posts-review to make sure that this report is ultimately relevant to the final published source code of the Lien team.

Mainnet Launch Verification

We were asked by the Lien team to verify the team's mainnet launch and correlate the public contract address bytecodes and respective source code with the code that was audited as part of this report.

We certify that the deployed bytecode of each of the following addresses corresponds to the linked GitHub contracts:

- MarketOracle: GitHub Link of Contract
- LienToken: GitHub Link of Contract
- BondTokenName: GitHub Link of Contract
- LbtPrice (LBTPricing): GitHub Link of Contract
- BondMaker: GitHub Link of Contract
- StableCoin: GitHub Link of Contract
- AuctionBoard: GitHub Link of Contract
- Auction: GitHub Link of Contract
- DecentralizedOTC: GitHub Link of Contract
- Wrapper: GitHub Link of Contract
- PriceCalculator: GitHub Link of Contract
- SpreadCalculator: GitHub Link of Contract
- IDOLvsLienExchange (LienBoxExchange): GitHub Link of Contract
- IDOLvsETHExchange: GitHub Link of Contract
- IDOLvsLBTExchangeFactory (LBTExchangeFactory): GitHub Link of Contract
- ChainlinkPriceOracle: GitHub Link of Contract
- TrustedPriceOracle: GitHub Link of Contract

We should note that minor differences exist between the BondTokenName and LBTPricing verified code and GitHub code, however these are linter styles that were previously un-applied and thus do not affect the bytecode.

Additionally, the AuctionBoard contract contains some differences between itself and its GitHub version namely due to flattening as additional keywords were necessitated for Etherscan to properly verify the contract. This difference does not affect the logic of the contract and thus we can verify that it is logically equivalent to the source code of the GitHub repository barring for any Solidity compilation bugs.

Findings

iDOL Repository

The iDOL repository contains the main implementation of the Lien protocol, particularly the auction processes, helper mechanisms, iDOL derivative backed token and overall the core premise of the protocol. Additionally, a lot of utility contracts are included in this repository and are meant to act as helper submodules of the main solutions.

DecentralizedOTC.sol (OTC)

The decentralized over-the-counter market implementation of Lien that enables participants to create pools of ERC20 tokens with their respective price calculator oracles. Contains logic that calculates the exchange rates between an LBT and ERC20 pair as well as configuration methods to properly set up the corresponding pools.

OTC- 01

TITLE	TYPE	SEVERITY	LOCATION
Visibility & Mutability Specifiers Missing	Coding Style	Informational	DecentralizedOTC.sol#L14-L15

[INFORMATIONAL] Description:

The aforementioned variables have been declared yet lack a visibility specifier. Additionally, their state mutability can be restricted to `immutable`.

Recommendations:

We advise that these traits are properly set.

Alleviation:

The Lien team properly set the visibility and mutability specifiers of each variable.

OTC-02

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `if` Block	Optimization	Informational	DecentralizedOTC.sol#L175-L179

[INFORMATIONAL] Description:

The comparison conducted on the `else` branch of the block should be set to a strict less-than (`<`) comparison instead of a less-than-or-equal (`<=`) comparison as the equality would result in the value zero being assigned to `n` which is its already set default value.

Additionally, the current block conducts 4 external function invocations in a best-case scenario and 8 external function invocations in a worst-case scenario as the functions `decimals()` are dynamically evaluated within the comparisons as well as the calculations. We advise that the results of these invocations are instead stored within in-memory variables that are subsequently utilized for the comparisons and assignments.

Recommendations:

Included in the description above.

Alleviation:

All our recommendations were assimilated whereby the conditional was properly adjusted and an in-memory variable was declared for the `decimals()` invocations.

OTC-03

TITLE	TYPE	SEVERITY	LOCATION
Error Rephrasal	Coding Style	Informational	DecentralizedOTC.sol#L180

[INFORMATIONAL] Description:

The current error message should be rephrased to state that the "decimal gap needs to be..." rather than "decimals needs to be...". Additionally, the rationale behind a 18 decimal gap difference maximum should be documented as the consequent multiplication would overflow under extreme and non-blocking circumstances.

Recommendations:

We advise the error message is rephrased and a comment is inserted preceding it that details the above warning.

Alleviation:

The error message was adjusted and a comment was inserted above the `require` statement.

OTC-04

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	DecentralizedOTC.sol#L197, L235, L266, L280

[INFORMATIONAL] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The linked comparisons were properly set to their inequality counterpart.

OTC-05

TITLE	TYPE	SEVERITY	LOCATION
Redundant `else` Blocks	Ineffectual Code	Informational	DecentralizedOTC.sol#237-L239 & L269-L271

[INFORMATIONAL] Description:

Both `else` blocks assign the value literal zero to a variable that already has the said value set as its default value.

Recommendations:

We advise that these `else` blocks are omitted.

Additionally, the declaration of L232 should be moved within the `if` block of L233 to L246 as it is not being used outside of it.

Alleviation:

Our recommendations were applied on the codebase to the letter.

OTC-06

TITLE	TYPE	SEVERITY	LOCATION
Unreachable Error Message	Ineffectual Code	Informational	DecentralizedOTC.sol#L288

[INFORMATIONAL] Description:

The only circumstance this error message would be emitted by the `_transferETH` implementation is in the scenario that the sender does not possess the necessary balance.

Recommendations:

As the actual balance of the contract is directly passed to the function invocation on L287, the function will not emit the error under any circumstance thus rendering it unreachable.

Alleviation:

The error message was omitted from the function invocation correctly.

LBTPricing.sol (LBP)

This contract utilizes Taylor expansion as well as other formulas defined in the whitepapers of Lien to conduct the pricing calculations of the various liquid bond tokens.

LBP-01

TITLE	TYPE	SEVERITY	LOCATION
Visibility Specifiers Missing	Coding Style	Informational	ERC20RegularlyRecord.sol#L33

[INFORMATIONAL] Description:

The aforementioned variables have been declared as `immutable` yet lack a visibility specifier. We advise that their visibility is properly set.

Recommendations:

We advise that their visibility is properly set.

Alleviation:

Their visibility specifiers were strictly set nullifying this Exhibit.

AuctionSecret.sol (AUS)

A basic getter-setter implementation of a storage structure that associates an auction ID as well as the hash of multiple bids and a secret to `Secret`, i.e. the bidder, amount and amount of iDOL.

AUS-01

TITLE	TYPE	SEVERITY	LOCATION
Duplicate Getter Function	Ineffectual Code	Informational	AuctionSecret.sol#L11 & L39-L41

[Informational] Description:

The contract variable `auctionSecret` is declared as `public`, thus having a compiler-generated getter, whilst a user-defined getter is also defined between L39 and L41 titled `_getSecret`.

Recommendations:

We advise that one of those getter functions is omitted to reduce the bytecode of the contract.

Alleviation:

The user-defined getter function was omitted from the codebase.

BondToken.sol (BDT)

A contract implementation of a simple mintable bond token whereby burning its value withdraws any ETH stored within the contract to the burner.

BDT-01

TITLE	TYPE	SEVERITY	LOCATION
Mutability Specifier Missing	Coding Style	Informational	BondToken.sol#L9

[INFORMATIONAL] Description:

The aforementioned variable(s) are only assigned to once during the construction of the contract and as such can be declared as `immutable` to reduce the gas cost of the code that utilizes them.

Recommendations:

We advise that their state mutability is set to “immutable”.

Alleviation:

Their state mutability was properly set to “immutable”.

BDT-02

TITLE	TYPE	SEVERITY	LOCATION
Function Optimization	Optimization	Informational	BondToken.sol#L63-L81

[INFORMATIONAL] Description:

The function `expire` can be significantly optimized and reduced in no. of lines by assigning the result of `isExpired()` to the `firstTime` variable, checking whether it is `false` and in such a case calling `_setRate` and finally emitting `LogExpire` with the final argument being the `firstTime` variable.

Recommendations:

Included in the description above.

Alleviation:

Our recommendation was fully assimilated in the codebase.

BDT-03

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual Comparisons	Ineffectual Code	Informational	BondToken.sol#L140-L143

[INFORMATIONAL] Description:

The first comparison of the `require` statement can instead be converted to a more efficient inequality with zero as the `rate.denominator` variable is an unsigned integer, whereas the second conditional will never fail as `rate.numerator` is also an unsigned integer and as such can only represent non-negative values.

Recommendations:

Included in the description above.

Alleviation:

Our recommendation was fully assimilated in the codebase.

BDT-04

TITLE	TYPE	SEVERITY	LOCATION
Trailing Ether	Mathematical	Minor	BondToken.sol#L100-L102

[INFORMATIONAL] Description:

The fractional calculation carried in the aforementioned lines defines how much Ether corresponds to a `burn` of a particular `amount` of the bond token. As the calculation is fractional, Ether remainders can possibly be locked within the contract forever.

Recommendations:

We advise that a remainder mechanism or a conditional that checks whether the full supply of the bond token has been burned and subsequently transmits any remainder to either the constructor of the contract or the last withdrawee.

Alleviation:

The Lien team took note of this Exhibit and, after weighing the trade-offs of implementing a gas-expensive workaround vs. the actual value of properly accounted for funds, decided to retain the current implementation as is.

BondTokenName.sol (BTN)

A utility contract that exposes methods for constructing the long and short name of a bond token.

BTN-01

TITLE	TYPE	SEVERITY	LOCATION
Tight-Packing Notice	Ineffectual Code	Informational	BondTokenName.sol#L30-L34

[INFORMATIONAL] Description:

The current implementation poses no issues, however we would like to note something for a potential update in the codebase. When the function `abi.encodePacked` is invoked with array arguments, its invocation with the following arrays would result in the same output:

```sol
string memory monthA = "1";
string memory dayA = "12";
string memory monthB = "11";
string memory dayB = "2";
assert(
  keccak256(abi.encodePacked(monthA, dayA)) == keccak256(abi.encodePacked(monthB, dayB))
);
```

```

**Recommendations:**

As such, we advise that `abi.encodePacked` invocations are carefully evaluated and set to `abi.encode` where redundant.

**Alleviation:**

The Lien team took note of our recommendation and will evaluate it in any subsequent Solidity development they carry out.
```

DecimalSafeMath.sol (DSM)

A utility contract that exposes methods for carrying out calculates with decimal places.

DSM-01

TITLE	TYPE	SEVERITY	LOCATION
Unsafe Multiplications	Mathematical	Major	DecimalSafeMath.sol#L7 & L16

[MAJOR] Description:

Two unsafe multiplications are performed within the function bodies of `decimalDiv` and `decimalMul` respectively which should be wrapped and safely carried out as they can potentially overflow.

Recommendations:

We advise that these computations are carried out safely as the name of the library implies.

Alleviation:

The Lien team informed us that this is merely a development dependency and will not be utilized in the final contracts, thus they opted not to fix the issue.

UseSafeMath.sol (USM)

A utility contract that wraps several built-in numerical types of Solidity with the safe-counterpart math libraries from OpenZeppelin.

No findings were found in this implementation.

UseOracle.sol (UOR)

A utility contract that exposes methods for managing the contract's oracle and retrieving data from it.

UOR-01

TITLE	TYPE	SEVERITY	LOCATION
Mutability Specifier Missing	Optimization	Informational	UseOracle.sol#L8

[INFORMATIONAL] Description:

The aforementioned variable(s) are only assigned to once during the construction of the contract and as such can be declared as `immutable` to reduce the gas cost of the code that utilizes them.

Recommendations:

We advise that their state mutability is set to “immutable”.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

UOR-02

TITLE	TYPE	SEVERITY	LOCATION
`require` to Modifier	Optimization	Informational	UseOracle.sol#L25 & L42

[INFORMATIONAL] Description:

The aforementioned `require` statements can be grouped into modifier(s) that increase the legibility and maintainability of the codebase.

Recommendations:

We advise that they are instead imposed via a `modifier` to aid in the maintainability of the codebase.

Alleviation:

The `require` statements were instead removed from the functions as they are `internal` and their limitation can be imposed by the contracts that utilize it.

SolidBondSafety.sol (SBS)

A contract that carries out the Black-Scholes formula approximations of the minimum threshold acceptable strike price relative to the current oracle price for an SBT based on data retrieved from the Oracle.

SBS-01

TITLE	TYPE	SEVERITY	LOCATION
Documentation Discrepancy	Comment	Informational	SolidBondSafety.sol#L26-L30

[INFORMATIONAL] Description:

The body of the function does not properly conduct the comparisons detailed in the aforementioned lines of comments as the first member of the `x` comparison should be a strict less-than (`<`) comparison rather than a less-than-or-equal (`<=`) comparison, i.e. L26 `0.3576 <= x <= 0.7751` should be set to `0.3576 < x <= 0.7751`.

Recommendations:

We advise that the comment line is properly adjusted to reflect the true cases evaluated within.

Alleviation:

The comment was corrected to reflect the right type of comparisons.

DateTimeLibrary.sol (DTL)

A utility contract that includes methods for converting unix timestamps to human-readable representations and vice-versa.

As it is an officially recognized library, it was not part of the audit scope.

Digits.sol (DGT)

A utility library that converts a `uint256` to its `string` representation according to the input digit precision.

DGT-01

TITLE	TYPE	SEVERITY	LOCATION
Input Variable Utilization	Optimization	Informational	Digits.sol#L8-L10

[INFORMATIONAL] Description:

The function `toString` can directly utilize the input variables instead of re-assigning them to new in-memory variables on L9 and L10.

Recommendations:

We advise that the input variables of the function are directly utilized.

Alleviation:

The input variables are directly utilized by the `toString` function in its latest iteration.

Polyline.sol (PLL)

A utility contract that exposes methods for handling line and line segment calculations using `x` and `y` coordinates for the maps defined in `BondMaker`.

No findings were found in this implementation.

Time.sol (TIM)

A utility contract that exposes a single method for retrieving the current unix timestamp on the blockchain.

No findings were found in this implementation.

TransferETH.sol (TEH)

A utility contract that exposes methods for handling the outgoing transfer of Ether from a contract.

No findings were found in this implementation.

Auction.sol (AUC)

The main implementation of a decentralized Vickrey auction whereby bids are secret. Contains all the relevant calculations for handling the full lifecycle of the auction.

AUC-01

TITLE	TYPE	SEVERITY	LOCATION
Visibility Specifiers Missing	Ineffectual Code	Informational	Auction.sol#L26, L27, L28, L30, L32-L35 & L57-L60

[INFORMATIONAL] Description:

The aforementioned variables have been declared yet lack a visibility specifier.

Recommendations:

We advise that their visibility is properly set.

Alleviation:

Their visibility was strictly set according to our recommendations were applicable.

AUC-02

TITLE	TYPE	SEVERITY	LOCATION
Redundant Duplicate `mapping`s	Ineffectual Code	Informational	Auction.sol#L37-L60

[INFORMATIONAL] Description:

All the `mapping` declarations contained in the above code segment can be grouped into a single `mapping` declaration that points to a `struct` containing the variable types of all mappings in a single structure.

Recommendations:

We advise this pattern is followed to reduce the lookup cost of the values as well as the gas cost of interacting with them.

If this Exhibit is not applied, we also advise that the variable types of L45, L50, L55 and L60 are adjusted to fit a full 32-byte (256-bit) slot as their current structure is inefficient and actually costs more gas to interact with.

Alleviation:

This pattern was utilized for almost all variables except the `auctionID2BondID` mapping, which ultimately is sensible to be independent from an auction’s configuration parameters.

AUC-03

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Data Type	Optimization	Informational	Auction.sol#L32-L35

[INFORMATIONAL] Description:

The data type of the `_bondIDAuctionCount` mapping is a `uint16`.

Recommendations:

As the EVM is geared towards 32-byte data types, it costs more gas to interact with and utilize a `uint16` variable than a `uint256`. As such, we advise that this is instead set to a functionally identical `uint256`.

Alleviation:

The variable type was properly adjusted to a `uint256`.

AUC-04

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Comparison	Optimization	Informational	Auction.sol#L163 & L235

[INFORMATIONAL] Description:

The aforementioned `if` blocks compare the iterator of the `for` loop to check whether this is not the first iteration of the loop.

Recommendations:

As such, the comparison can instead be converted to an inequality with zero as the iterator begins at zero and is incremented by two on each iteration.

Alleviation:

The linked comparisons were properly set to inequality w/ zero conditionals.

AUC-05

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	Auction.sol#L100, L328 & L405

[INFORMATIONAL] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were changed to inequality comparisons apart from the last one which was completely omitted as the function is only executed by the StableCoin contract which in turn handles zero values gracefully.

AUC-06

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `if` Blocks	Optimization	Informational	Auction.sol#L180-L188 & L250-L272

[INFORMATIONAL] Description:

These `if` blocks can be optimized by moving the `require` statements within `else` blocks of the `if` clauses and adjusting the `require` comparisons to not check duplicatively for the comparison done in the `if` block.

Additionally, within finding A it is possible for the calculation of L188 to precede the full `if` block and simply conduct the subtraction of L184 within the inner-most `if` block.

Recommendations:

Included in the description above.

Alleviation:

Our recommendations were applied in full on the codebase.

AUC-07

TITLE	TYPE	SEVERITY	LOCATION
Ambiguous Variable Name	Coding Style	Informational	Auction.sol#L213

[INFORMATIONAL] Description:

The variable name `myLowestVerify` should be documented or set to something more descriptive, similarly within `AuctionBoard`.

Recommendations:

We advise that supplementary comments accompany the variable declaration and / or utilization to clearly depict its purpose.

Alleviation:

Such documentation was inserted where applicable for the `myLowestVerify` variable.

AUC-08

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `if` Block	Optimization	Informational	Auction.sol#L372-L382

[INFORMATIONAL] Description:

The current `if` clause structure could be adjusted to completely omit the last `else` block and negate the need for a conditional on the second `else` block by moving the `assert` of L381 to precede the `if` block and adjusting it to a `require`.

Recommendations:

Included in the description above.

Alleviation:

Our recommendation was assimilated in full within the codebase.

AUC-09

TITLE	TYPE	SEVERITY	LOCATION
Checks Effects Pattern	Coding Style	Informational	Auction.sol#L385-L386

[INFORMATIONAL] Description:

The reset of the participant's info should occur prior to the transfer of iDOL.

Recommendations:

This particular instance poses no vulnerability as no re-entry is possible via iDOL, however it is a security practise to employ the Checks Effects pattern.

Alleviation:

The Checks Effects pattern was properly applied in the linked code segment.

AUC-10

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `if` Block	Optimization	Informational	Auction.sol#L421-L428

[INFORMATIONAL] Description:

The aforementioned `if` block can be optimized whereby the `_publishSettledAverageAuctionPrice` function is invoked outside the `if` clause with the last argument being `isLast`, rendering the `if` block to only need a single clause that executes L427 if `isLast` is `false`.

Recommendations:

Included in the description above.

Alleviation:

Our recommendation was applied in full on the codebase.

AUC-11

TITLE	TYPE	SEVERITY	LOCATION
Redundant `abi.encodePacked` Invocation	Ineffectual Code	Informational	Auction.sol#L558

[INFORMATIONAL] Description:

The arguments passed to `abi.encodePacked` both occupy a full word (32-bytes) and as such, the `abi.encodePacked` result will be equivalent to the `abi.encode` result.

Recommendations:

We advise that the `abi.encode` counterpart is utilized instead.

Alleviation:

The `abi.encode` variant is being properly utilized in the linked line.

AUC-12

TITLE	TYPE	SEVERITY	LOCATION
Redundant Encoding & Decoding	Ineffectual Code	Informational	Auction.sol#L713

[INFORMATIONAL] Description:

The statement `abi.decode(abi.encode(auctions[i]), (uint256))` is equivalent to `uint256(auctions[i])`.

Recommendations:

We advise that type casting is utilized instead.

Alleviation:

The statement was properly set to its optimized equivalent detailed above.

AuctionBoard.sol (AUB)

Contains the core logic for the lifecycle of bids in a particular auction, including all types of utility methods such as sorting the bids of an auction.

AUB-01

TITLE	TYPE	SEVERITY	LOCATION
Visibility Specifiers Missing	Coding Style	Informational	AuctionBoard.sol#L23, L24, L25, L26, L27, L28, L30, L31, L32, L33, L34, L36, L37, L38, L39, L41, L64, L69, L82 & L83

[INFORMATIONAL] Description:

The aforementioned variables have been declared yet lack a visibility specifier.

Recommendations:

We advise that their visibility is properly set.

Alleviation:

Their visibility specifiers were strictly set.

AUB-02

TITLE	TYPE	SEVERITY	LOCATION
Redundant Duplicate `mapping`s	Ineffectual Code	Informational	AuctionBoard.sol#L55, L64, L69, L80, L82, L83, L98, L112 & L118

[INFORMATIONAL] Description:

All the `mapping` declarations contained in the above code segment can be grouped into a single `mapping` declaration that points to a `struct` containing the variable types of all mappings in a single structure.

Recommendations:

We advise this pattern is followed to reduce the lookup cost of the values as well as the gas cost of interacting with them.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

AUB-03

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Data Types	Optimization	Informational	AuctionBoard#L64, L69, L82 & L83

[INFORMATIONAL] Description:

These variables utilize `uint64` keys for two `mapping`s and also set `uint64` arrays as the result of two `mapping` lookups.

Recommendations:

As the EVM is designed to work with 32-byte variables, it is more optimal to utilize `uint256` across the board to reduce the gas cost of all statements that affect these variables.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

AUB-04

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	AuctionBoard.sol#L262, L355, L356, L373, L547, L831

[INFORMATIONAL] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were properly set to their inequality counterparts.

AUB-05

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `struct` Variable Types	Optimization	Informational	AuctionBoard.sol#L49-L54, L60-L63, L76-L79, L91-L97 & L114-L118

[INFORMATIONAL] Description:

Within the EVM, it is optimal to set the variable types of a `struct` in such an order that they are packed into 256-bit slots neatly. The aforementioned data structures do not utilize the full 256-bit space of each slot and as such lead to the generated bytecode needing to conduct padding operations and subsequently cost more to carry out.

Recommendations:

We advise that the variables are adjusted to properly utilize the total bits necessary to fill a full slot.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

AUB-06

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual `require` Statement	Ineffectual Code	Informational	AuctionBoard.sol#L300

[INFORMATIONAL] Description:

This `require` statement will never throw as `priceCount` is of type `uint64` and was safecasted in the preceding line.

Recommendations:

We advise that the `require` statement is omitted.

Alleviation:

The SafeCast operation was omitted instead nullifying this Exhibit.

AUB-07

TITLE	TYPE	SEVERITY	LOCATION
`if` Block to `require`	Coding Style	Informational	AuctionBoard.sol#L302-L304

[INFORMATIONAL] Description:

This `if` block checks a conditional that, when evaluated to `true`, will halt the transaction's execution with a `revert`.

Recommendations:

We advise that this is changed to a `require` instead.

Additionally, the word `MAX` within a `constant` variable is utilized inclusive for `MAX_UINT64` and exclusive for `MAX_BOARD_NUM`. Proper conditionals should be evaluated for these variables.

Alleviation:

Our recommendation was assimilated in full within the codebase.

AUB-08

TITLE	TYPE	SEVERITY	LOCATION
`else` Clause to `require`	Coding Style	Informational	AuctionBoard.sol#L435-L442

[INFORMATIONAL] Description:

The final `else` block of the `if` conditional will halt the transaction's execution with a `revert`.

Recommendations:

We advise that this is changed to a `require` instead that precedes the `if` block, with the `if` block's conditionals properly adjusted.

Alleviation:

An equivalent modification was carried out in the codebase simplifying the logic of the linked code blocks.

AUB-09

TITLE	TYPE	SEVERITY	LOCATION
Conditional Optimization	Optimization	Informational	AuctionBoard.sol#L464-L466

[INFORMATIONAL] Description:

This `if` block should be moved to the beginning of the function to ensure no redundant statements are executed.

Recommendations:

Included in the description above.

Alleviation:

The `if` block was properly moved to the beginning of the function.

AUB-10

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual `break` Statement	Ineffectual Code	Informational	AuctionBoard.sol#L535

[INFORMATIONAL] Description:

This `break` statement is executed during the last iteration of the `for` loop and as such, it is redundant.

Recommendations:

We advise that it is safely omitted.

Alleviation:

The `break` statement was properly removed from the last iteration of the `for` loop.

AuctionTimeControl.sol (ATC)

Contains utility methods for handling the status of an auction as well as verifying it within conditionals.

ATC-01

TITLE	TYPE	SEVERITY	LOCATION
Visibility Specifiers Missing	Coding Style	Informational	AuctionTimeControl.sol#L7, L8, L9, L10, L11, L12, L14, L15, L16, L17, L18 & L34

[INFORMATIONAL] Description:

The aforementioned variables have been declared yet lack a visibility specifier.

Recommendations:

We advise that their visibility is properly set.

Alleviation:

A visibility specifier was strictly set for the linked declarations.

ATC-02

TITLE	TYPE	SEVERITY	LOCATION
`constant` to `enum`	Coding Style	Informational	AuctionTimeControl.sol#L14-L18

[INFORMATIONAL] Description:

The variables of the aforementioned block could potentially be converted to a functionally equivalent `enum` instead.

Recommendations:

Included in the description above.

Alleviation:

The variables were properly converted to an `enum` increasing the legibility and maintainability of the codebase.

ATC-03

TITLE	TYPE	SEVERITY	LOCATION
Redundant Duplicate `mapping`s	Ineffectual Code	Informational	AuctionTimeControl.sol#L20-L23 & L25-L29

[INFORMATIONAL] Description:

All the `mapping` declarations contained in the above code segment can be grouped into a single `mapping` declaration that points to a `struct` containing the variable types of all mappings in a single structure.

Recommendations:

We advise this pattern is followed to reduce the lookup cost of the values as well as the gas cost of interacting with them.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

ATC-04

TITLE	TYPE	SEVERITY	LOCATION
Inconsistent Calculations	Mathematical	Informational	AuctionTimeControl.sol#L112-L117

[INFORMATIONAL] Description:

The calculation carried out in those lines rounds off the trailing unix seconds left to ensure that the resulting `closingTime` is a multiple of an hour. However, the `closingTime` calculation on L113 does not round off the final `closingTime` value

Recommendations:

We advise that a similar level of granularity is also imposed on the `closingTime` calculation of emergency auctions.

Alleviation:

A granularity of 5 minutes was imposed for emergency auctions.

BondMaker.sol (BMK)

This contract implementation provides utility methods for creating issuing new bonds, interacting with the Polyline implementation for price calculations as well as other conversion methods such as from bonds to ETH.

BMK-01

TITLE	TYPE	SEVERITY	LOCATION
Visibility Specifiers Missing	Coding Style	Informational	BondMaker.sol#L22, L24 & L26

[INFORMATIONAL] Description:

The aforementioned variables have been declared yet lack a visibility specifier.

Recommendations:

We advise that their visibility is properly set.

Alleviation:

Strict visibility specifiers were set for the linked variable declarations.

BMK-02

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `struct` Variable Type	Optimization	Informational	BondMaker.sol#L31

[INFORMATIONAL] Description:

Within the EVM, it is optimal to set the variable types of a `struct` in such an order that they are packed into 256-bit slots neatly. The aforementioned data structures do not utilize the full 256-bit space of each slot and as such lead to the generated bytecode needing to conduct padding operations and subsequently cost more to carry out.

Recommendations:

We advise that the specified variable is set to a `uint256` as it already occupies a full word slot in the struct.

Alleviation:

The linked variable was properly set to a 256-bit size.

BMK-03

TITLE	TYPE	SEVERITY	LOCATION
Undocumented Fee Calculation	Coding Style	Informational	BondMaker.sol#L229

[INFORMATIONAL] Description:

The fee calculation at this point multiplies the value by 2 and divides it by 1002. The rationale behind this calculation should be properly documented within the codebase.

Recommendations:

We advise appropriate documentation is added for this calculation.

Alleviation:

Precise documentation was supplemented for this calculation.

BMK-04

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `mapping` Lookup	Optimization	Informational	BondMaker.sol#L350-L368

[INFORMATIONAL] Description:

As all members of the `struct` are accessed, the `mapping` lookup should instead be stored to a `memory` declaration of the `struct` whose members are subsequently assigned to the return variables.

Recommendations:

Included in the description above.

Alleviation:

The lookups were properly set to be stored in `memory` declarations instead of `storage` declarations.

StableCoin.sol (STC)

The iDOL derivative backed stablecoin implementation that interacts with derivative auctions. Contains core logic of a token as well as the corresponding auction-based minting mechanisms w/ solid bond tokens.

STC-01

TITLE	TYPE	SEVERITY	LOCATION
Visibility Specifiers Missing	Coding Style	Informational	StableCoin.sol#L27, L32, L33, L35, L36, L38 & L59

[INFORMATIONAL] Description:

The aforementioned variables have been declared yet lack a visibility specifier.

Recommendations:

We advise that their visibility is properly set.

Alleviation:

A strict visibility specifier was set for each contract level declaration.

STC-02

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `struct` Variable Types	Optimization	Informational	StableCoin.sol#L42-L50 & L66-L71

[INFORMATIONAL] Description:

Within the EVM, it is optimal to set the variable types of a `struct` in such an order that they are packed into 256-bit slots neatly. The aforementioned data structures do not utilize the full 256-bit space of each slot and as such lead to the generated bytecode needing to conduct padding operations and subsequently cost more to carry out.

Recommendations:

We advise that the variables are adjusted to properly utilize the total bits necessary to fill a full slot.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

STC-03

TITLE	TYPE	SEVERITY	LOCATION
Unreachable Error Message	Ineffectual Code	Informational	StableCoin.sol#L476

[INFORMATIONAL] Description:

The only circumstance this error message would be emitted by the `div` implementation is in the scenario that the divider is zero.

Recommendations:

As this statement cannot be executed when the divisor is zero, this error message is unreachable and thus redundant.

Alleviation:

The explicit error message was removed from the relevant code.

STC-04

TITLE	TYPE	SEVERITY	LOCATION
Redundant Multiplication & Divisions	Ineffectual Code	Informational	StableCoin.sol#L133, L290, L498 & L502

[INFORMATIONAL] Description:

The value of `LOCK_POOL_BORDER` is utilized for multiple multiplications and divisions throughout the codebase.

Recommendations:

As it is a `constant` set to the value `1`, all those statements can be safely omitted.

Alleviation:

All statements were properly omitted from the codebase.

UseAuction.sol (UAU)

Utility contract that allows the storage of an `Auction` address. Overall, all `Use` prefixed contracts could inherit from a common base that contains a `deployer` variable and a `modifier` for checking against it.

UAU-01

TITLE	TYPE	SEVERITY	LOCATION
Mutability Specifier Missing	Coding Style	Informational	UseAuction.sol#L7

[INFORMATIONAL] Description:

The aforementioned variable(s) are only assigned to once during the construction of the contract and as such can be declared as `immutable` to reduce the gas cost of the code that utilizes them.

Recommendations:

Included in the description above.

Alleviation:

The aforementioned variable was omitted from the contract thus rendering this exhibit irrelevant.

UseBondMaker.sol (UBM)

Utility contract that allows the storage of a `BondMaker` address.

UBM-01

TITLE	TYPE	SEVERITY	LOCATION
Mutability Specifier Missing	Coding Style	Informational	UseBondMaker.sol#L7

[INFORMATIONAL] Description:

The aforementioned variable(s) are only assigned to once during the construction of the contract and as such can be declared as `immutable` to reduce the gas cost of the code that utilizes them.

Recommendations:

Included in the description above.

Alleviation:

The aforementioned variable was omitted from the contract thus rendering this exhibit irrelevant.

UseBondTokenName.sol (UBT)

Utility contract that allows the storage of a `BondTokenName` address.

UBT-01

TITLE	TYPE	SEVERITY	LOCATION
Mutability Specifier Missing	Coding Style	Informational	UseBondTokenName.sol# L7

[INFORMATIONAL] Description:

The aforementioned variable(s) are only assigned to once during the construction of the contract and as such can be declared as `immutable` to reduce the gas cost of the code that utilizes them.

Recommendations:

Included in the description above.

Alleviation:

The aforementioned variable was omitted from the contract thus rendering this exhibit irrelevant.

UseExchangeLBTAndIDOLFactory.sol (ULI)

Utility contract that allows the storage of an `ExchangeFactory` address.

ULI-01

TITLE	TYPE	SEVERITY	LOCATION
Mutability Specifier Missing	Coding Style	Informational	UseBondTokenName.sol# L7

[INFORMATIONAL] Description:

The aforementioned variable(s) are only assigned to once during the construction of the contract and as such can be declared as `immutable` to reduce the gas cost of the code that utilizes them.

Recommendations:

Included in the description above.

Alleviation:

The aforementioned variable was omitted from the contract thus rendering this exhibit irrelevant.

UseStableCoin.sol (USC)

Utility contract that allows the storage of a `StableCoin` address.

USC-01

TITLE	TYPE	SEVERITY	LOCATION
Mutability Specifier Missing	Coding Style	Informational	UseStableCoin.sol#L7

[INFORMATIONAL] Description:

The aforementioned variable(s) are only assigned to once during the construction of the contract and as such can be declared as `immutable` to reduce the gas cost of the code that utilizes them.

Recommendations:

Included in the description above.

Alleviation:

The aforementioned variable was omitted from the contract thus rendering this exhibit irrelevant.

Wrapper.sol (WRP)

A wrapper contract that inherits from all `Use` prefixed contracts and provides utility methods to interact with them.

WRP-01

TITLE	TYPE	SEVERITY	LOCATION
Redundant Duplicate `mapping`s	Ineffectual Code	Informational	Wrapper.sol#L28 & L29

[INFORMATIONAL] Description:

As both are associated with a particular pool ID, it would be more optimal to instead utilize a single `mapping` that associates a pool ID with its total as well as an additional `mapping` that associates an `address` with the `amount` it has deposited to the pool.

Recommendations:

Included in the description above.

Alleviation:

This exhibit is no longer relevant as the linked code segments have been removed.

WRP-02

TITLE	TYPE	SEVERITY	LOCATION
Ineffectual `if` Clause	Ineffectual Code	Informational	Wrapper.sol#L85-L87 & L100-L102

[INFORMATIONAL] Description:

These `if` clauses assign the default value of a `uint256` to the variable `toBack` which already possesses that value.

Recommendations:

As such, it is possible to completely omit these `if` clauses.

Alleviation:

This exhibit is no longer relevant as the linked code segments have been removed.

Lien Token Repository

The Lien Token repository contains the implementaiton of the LIEN ERC-20 token with two additional functionalities, namely a vesting and a dividend mechanism. The codebase was evaluated with regards to its conformity to the ERC-20 specification as well as its correct and optimal implementation of the logic it is meant to fulfill.

ERC20Vestable.sol (ERV)

This file is a custom extension of the ERC-20 specification that enables the creation of "Grants" with durations for a particular address that releases funds based on the duration that has elapsed from the Grant's creation date and end time.

ERV-01

TITLE	TYPE	SEVERITY	LOCATION
Unlocked Compiler Version	Language Specific	Informational	ERC20Vestable.sol#L1

[INFORMATIONAL] Description:

An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers.

This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendations:

We advise that the compiler version is instead locked at the lowest version possible that the full project can be compiled at. **This is evident across the codebase on multiple contracts and will only be mentioned once, however it applies to all**.

Alleviation:

The compiler version was locked by getting rid of the `^` version specifier in the `pragma` statement thus locking the contract's compiler version at `0.6.5`.

ERV-02

TITLE	TYPE	SEVERITY	LOCATION
Ambiguous Comment Explanation	Coding Style	Informational	ERC20Vestable.sol#L16

[INFORMATIONAL] Description:

Line 16 details that additionally deposited tokens to an already started grant "increase the amount of the vesting". Our initial assumption with this sentence was that the vesting duration would increase should a new deposit occur.

Recommendations:

We advise this to be rephrased to "increase the amount vested".

Alleviation:

The comment was rephrased to properly depict its purpose.

ERV-03

TITLE	TYPE	SEVERITY	LOCATION
Singular Form of Plural Variable	Coding Style	Informational	ERC20Vestable.sol#L39

[INFORMATIONAL] Description:

The variable `totalRemainingGrant` refers to the total remaining grants that have yet to be claimed by the individuals they are meant for.

Recommendations:

We advise that the plural form is utilized for the variable instead to increase the legibility of the codebase.

Alleviation:

The plural form of the variable name was properly set.

ERV-04

TITLE	TYPE	SEVERITY	LOCATION
Event Variable Order Inconsistency	Ineffectual Code	Informational	ERC20Vestable.sol#L41-L52

[INFORMATIONAL] Description:

The events `CreateGrant` and `DepositToGrant` contain similar variables that are indexed, however the way the variables are ordered within the event differ.

Recommendations:

We advise that `CreateGrant` is restructured so that the new variable order is `beneficiary`, `id`, `creator` and `endTime` conforming to how `DepositToGrant` and `ClaimVestedTokens` are structured.

Alleviation:

The event variable order was adjusted to be consistent between the two events.

ERV-05

TITLE	TYPE	SEVERITY	LOCATION
Incorrect Comment	Coding Style	Informational	ERC20Vestable.sol#L65

[INFORMATIONAL] Description:

The comment on L65 of the contract states that the `beneficiary` variable is the "receipt of vested tokens of the grant".

Recommendations:

The word "receipt" should be changed to "recipient".

Alleviation:

The comment was properly corrected to use the correct word.

ERV-06

TITLE	TYPE	SEVERITY	LOCATION
Struct Optimization	Optimization	Informational	ERC20Vestable.sol#L25-L26

[INFORMATIONAL] Description:

The `Grant` struct contains two members signaling the start and end of the grant, the `startTime` and `endTime` variables both of which are of the `uint256` type.

Recommendations:

As these are meant to represent Unix timestamps, they can safely be set to the `uint128` variable type.

Unix timestamps are traditionally represented by 32-bits with higher precision machines representing them with 64-bits. Thus, 128-bits are more than enough to represent a unix timestamp under the context of the project.

The reason lower-bit precision numbers are not suggested is that the EVM operates on a word-basis, meaning that it stores everything in 256-bit slots and as such by applying this optimization the `Grant` struct would be stored in 3 slots instead of 4, greatly reducing the gas cost of reading from and writing to storage.

Alleviation:

The latter two variables of the `Grant` struct were reduced to 128-bits of precision, reducing the overall gas cost involved with reading and writing `Grant` structs from and to storage.

ERV-07

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Utilization of `storage` Pointers	Optimization	Informational	ERC20Vestable.sol#L109-L121, L145-L162 & L206-L214

[INFORMATIONAL] Description:

In Solidity, a variable declared as `storage` will not be read from until explicitly stated so in the codebase. As the contract is meant to be operated on a public network, each operation on the storage of the contract costs a significant amount of gas and subsequently currency to the caller.

Recommendations:

The code segments above contain duplicate reads that could instead be bypassed by utilizing in-memory variables.

ERV-07A could store `g.claimed.add(amount)` to an in-memory variable as `g.claimed` is both stored to and read from on L112 and L117 respectively, whilst L117 will result in the aforementioned in-memory calculation already being carried out.

ERV-07B should be changed to instead retrieve the `Grant` to `memory` rather than `storage` for two reasons. Firstly, if ERV-06 is applied then the code as is would perform a duplicate full-word read for `startTime` and `endTime`. Lastly, the function `_vestedAmount` should utilize the variables directly or a `memory` struct rather than accept a pointer to `storage` as the current code segment will read all variables of the struct redundantly twice.

ERV-07C refers to the `_vestedAmount` function that should be refactored to utilize in-memory variables as per ERV-07B.

Alleviation:

Each sub-exhibit was dealt with according to our suggestions above.

ERV-08

TITLE	TYPE	SEVERITY	LOCATION
Disallow Redemption of Zero Amounts	Optimization	Informational	ERC20Vestable.sol#L109-L121

[INFORMATIONAL] Description:

A conditional check should be applied here that disallows vested token claims of zero value that result from multiple claim transactions being included in the same block.

Recommendations:

As this function performs expensive operations and multiple storage reads and writes, it would be beneficial to `require` the amount is not equal to zero as it would halt the execution of the function, preventing unnecessary gas from being utilized and prevent zero-value `ClaimVestedTokens` events from being emitted.

Alleviation:

Each sub-exhibit was dealt with according to our suggestions.

ERV-09

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	ERC20Vestable.sol#L192

[INFORMATIONAL] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

On a side-note, it is possible to omit the `require` check of L192 and instead conduct the assignment `id = id - 1` before the check of L193, adjusting it slightly to use a less-than (`<`) comparison rather than a less-than-or-equal (`<=`) comparison.

Alleviation:

Both our recommendations were assimilated within the codebase.

ERC20RegularlyRecord.sol (ERR)

This file is a custom extension of the ERC20Snapshot.sol OpenZeppelin library that intends to snapshot balances in intervals rather than per transaction using a term-based mechanism.

ERR-01

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	ERC20RegularlyRecord.sol#L33

[INFORMATIONAL] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

All comparisons were properly set to their inequality counterparts.

ERR-02

TITLE	TYPE	SEVERITY	LOCATION
Redundant Greater-Than-Or-Equal Comparison	Ineffectual Code	Informational	ERC20RegularlyRecord.sol#L53

[INFORMATIONAL] Description:

The SafeMath library's `sub` function internally conducts the exact same comparison carried out in the aforementioned line and as such, it can be safely omitted. If a custom error message is desired, it can be passed in as a third argument to the `sub` invocation as it is supported by the OpenZeppelin library.

Recommendations:

Included in the description above.

Alleviation:

The error message was properly set as a second argument to the `sub` invocation.

LienToken.sol (LIT)

This file represents the LIEN token implementation that derives from both ERC20Vestable.sol and ERC20RegularlyRecord.sol and is meant to represent the token implementation of the Lien protocol.

LIT-01

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Token / ETH Differentiation Logic	Optimization	Informational	N/A

[INFORMATIONAL] Description:

The current token implementation differs between ERC-20 tokens and the native currency of the ecosystem, Ether, by maintaining separate variables for each type. This substantially increases the bytecode of the contract, the cost of interacting with it as well as code duplication in relation to accessing and calculation logic.

Recommendations:

It is possible to instead utilize a "reserved" address for representing ETH-type actions. This is a common paradigm utilized by other projects in the space and this address could for example be the `0x0000000000000000000000000000000000000000` address or the `0xFFfFfFffFFfffFFfFFfFFFFFffFFFffffFfFFFfF` address.

Alleviation:

The contract was overhauled to utilize a reserved address (`0x0000000000000000000000000000000000000000`) as representing ETH-type actions, significantly reducing the gas cost of all functions of the contract as well as its final bytecode.

LIT-02

TITLE	TYPE	SEVERITY	LOCATION
Duplicate `currentTerm()` Calls	Ineffectual Code	Informational	LienToken.sol#L88-L89, L100 & L345-L350

[INFORMATIONAL] Description:

The function `currentTerm`'s result will remain the same across a single invocation.

Recommendations:

As such it is possible to store it to an in-memory variable that is subsequently used twice instead of invoking the function twice redundantly.

Alleviation:

An in-memory variable was properly utilized for these calls.

LIT-03

TITLE	TYPE	SEVERITY	LOCATION
Redundant Dynamic Loop Conditional Evaluation	Ineffectual Code	Informational	LienToken.sol#L111-L118, L136-L140, L222, L235, L254-L261 & L280-L284

[INFORMATIONAL] Description:

The aforementioned loops utilize the result of the `currentTerm` function invocation in the conditional. As loop conditionals are evaluated dynamically, this results in the `currentTerm` function being invoked on each execution of the loop whilst its result remains the same within the same invocation.

Recommendations:

As such, we advise that its result is stored outside the loop and an in-memory variable is utilized instead within the conditional to optimize the gas cost.

Alleviation:

An in-memory variable was properly utilized in the conditional of the for loops.

LIT-04

TITLE	TYPE	SEVERITY	LOCATION
Combined "profit" & "paid" Retrieval Function	Ineffectual Code	Informational	LienToken.sol#L152-L182

[INFORMATIONAL] Description:

As the `Asset` struct members `profit` and `paid` are usually used in pairs, it would be wise to create an `internal` function that retrieves both to avoid duplicate `mapping` lookups in functions like L223 and L236.

Recommendations:

Included in the description above.

Alleviation:

The result of the lookup was set to be stored to an in-memory variable thus negating this exhibit.

LIT-05

TITLE	TYPE	SEVERITY	LOCATION
Function `_addProfit` Return	Ineffectual Code	Informational	LienToken.sol#L79-L101

[INFORMATIONAL] Description:

The function `_addProfit` is utilized twice across the codebase within `settleProfit` and `settleProfitOfEth` and it precedes a lookup to the profit at the current term, which is what `_addProfit` affects.

Recommendations:

As such, it is possible to have `_addProfit` return the result of the addition to the caller to prevent the redundant lookup being done on each function.

Alleviation:

The function was removed altogether in favor of a direct calculation.

LIT-06

TITLE	TYPE	SEVERITY	LOCATION
Potential Dividend Remainder	Mathematical	Minor	LienToken.sol#L352-L362

[Minor] Description:

The function `_dividend` calculates the resulting dividend of a member by multiplying the profit by a user's balance and subsequently dividing it by the total supply of the token.

Recommendations:

It is possible that a small amount of profit will never be redeemable on each term checkpoint. As the division rounds off and does not factor in the remainder, small traces of currency will always be locked on each term which could potentially mount to a significant balance.

The ramifications of this trait should be evaluated carefully and fallback measures should be defined in case the Lien protocol team desires the tokens to be rescue-able.

Alleviation:

The Lien team took note of this Exhibit and, after weighing the trade-offs of implementing a gas-expensive workaround vs. the actual value of properly accounted for funds, decided to retain the current implementation as is.

LIT-07

TITLE	TYPE	SEVERITY	LOCATION
Out-of-Gas Exception on Multi-Term Redemptions	Mathematical	Minor	LienToken.sol#L103-L150 , L216-L240 & L242-L289

[Minor] Description:

The functions that rely on a loop iteration of the terms that have passed between the start of the contract's dividend mechanism or a user's last redemption and the end of the said mechanism can result in out-of-gas exhaustions.

Recommendations:

To alleviate this issue, we advise that an additional input variable is defined that dictates how many terms should be evaluated in a particular invocation.

For LIT-07A, the value until which the loop should iterate should be equal to `Math.min(i.add(terms), currentTerm())` if we consider `terms` to be the input number of terms.

For LIT-07B, the solution is difficult as the function is meant to iterate through all previously settled profits and redemptions. To solve this, a "timestamping" mechanism could be utilized whereby each new invocation begins from the last term the function was invoked from.

For LIT-07C, the same solution as LIT-07A could be applied.

Alleviation:

The gas ramifications of the for loops were closely examined and were deemed to be of a tolerable gas cost, meaning that no such additional mechanism is necessary as even under the worst case scenario the function would pose acceptable gas costs.

LIT-08

TITLE	TYPE	SEVERITY	LOCATION
Snapshot Mechanism Bypass	Ineffectual Code	Minor	LienToken.sol#L72

[Minor] Description:

When the Lien token is initially constructed, it bypasses the snapshotting mechanism implementations of the `_mint` function and invokes it directly from the `ERC20` implementation.

As the token is neither burnable nor mintable, the total supply of the token is never snapshotted and as such a significant amount of code is executed for no reason when retrieving the total supply at a particular term like L196 and L212, which subsequently lookup for a snapshot at the specified term that does not exist.

Ultimately, the function falls back to the default getter function of the total supply as evident on https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.0.0/contracts/token/ERC20/ERC20Snapshot.sol#L102 of the OpenZeppelin ERC20Snapshot library at v3.0.0.

Recommendations:

To avoid the redundantly executed statements, the `totalSupplyAtTermEnd` invocations should be replaced by direct invocations of the token's `totalSupply` unless minting and burning mechanisms are envisioned to be added to the token in a future version.

Alleviation:

The invocations were properly replaced with an optimized direct call to the token's `totalSupply`.

Oracle Repository

The Oracle repository contains the implementation of a contract that interfaces with other oracles, such as Chainlink, and not only provides the price of a particular asset but also the price's volatility as calculated based on the accompanying [Parameter Tuning White Paper] https://lien.finance/pdf/LienParameterTuningWP_v1.pdf by Lien.

The implementation utilizes a fallback system whereby if the primary decentralized oracle is detected to be malfunctioning, a fallback centralized oracle is utilized instead temporarily until the primary oracle returns to normal behavior or is replaced by one that is.

ChainlinkPriceOracle.sol (CPO)

This file contains a Chainlink-based oracle implementation that utilizes an underlying aggregator to retrieve the price of a particular asset.

CPO-01

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	ChainlinkPriceOracle.sol# L8

[INFORMATIONAL] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

On a side-note, the return variable of the function is named as `r` and can be omitted as an explicit `return` statement is utilized instead.

Alleviation:

The comparisons were set to their optimized counterparts correctly.

CPO-02

TITLE	TYPE	SEVERITY	LOCATION
Nested `try-catch` Block Optimization	Optimization	Informational	ChainlinkPriceOracle.sol# L90-L136

[INFORMATIONAL] Description:

The current implementation of `_isLatestPriceProper` contains `return` statements that can be avoided by naming the return variable.

Recommendations:

Should the return variable be named, the `return` statements of L130 and L133 are redundant whilst the return statement of L135 should be moved inside the innermost `try` block after the `if` block of L126-L128.

On a side-note, we would also advise that a comment line is inserted around the constant `FLUCTUATION_THRESHOLD` to indicate that it represents a eight-fold (800%) fluctuation window.

Alleviation:

The code blocks were properly optimized to the greatest extent possible. Additionally, our comment recommendation was also added preceding the `FLUCTUATION_THRESHOLD` variable declaration.

CPO-03

TITLE	TYPE	SEVERITY	LOCATION
Nested `try-catch` Block Optimization	Optimization	Minor	ChainlinkPriceOracle.sol# L106

[Minor] Description:

The function `_isLatestPriceProper` is meant to utilize multiple `try-catch` blocks to ensure that it can be safely executed under any circumstance surrounding the `aggregator` contract.

However, the subtract occuring on L106 can result in an underflow if `latestRound` is zero whereby an exception will be thrown by the SafeMath library and it will be uncaught, causing the whole function to error and not return anything.

Recommendations:

We advise that a conditional is imposed before the `try` statement of L106 that ensures `latestRound` is not equal to `0`.

Alleviation:

A proper conditional was imposed before the `try` statement.

TrustedPriceOracle.sol (TPO)

This contract is meant to implement a centralized oracle that would act as the fallback of `MarketOracle.sol` and whose owner will have totalitarian control over the price reported by it. Under normal circumstances, this oracle would not be utilized by the `MarketOracle.sol`.

TPO-01

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	TrustedPriceOracle.sol#L32, L46, L66, L74 & L110

[INFORMATIONAL] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The linked conditional statements were optimized and converted to their inequality counterparts.

TPO-02

TITLE	TYPE	SEVERITY	LOCATION
Double `mapping` to Struct	Ineffectual Code	Informational	TrustedPriceOracle.sol#L21-L24

[INFORMATIONAL] Description:

As the mappings `prices` and `timestamps` associate a single "id" with a corresponding price and timestamp, a single `mapping` could be utilized that instead maps a single "id" to a struct composed of those two variables.

Additionally, the struct could be optimized to fit in a single 256-bit slot by reducing the number of bits in the timestamp and price required if deemed possible.

Recommendations:

Included in the description above.

Alleviation:

A struct named `Record` was devised that holds those two variables and is the result of a mapping lookup as we advised.

TPO-03

TITLE	TYPE	SEVERITY	LOCATION
Redundant "OR" Clause	Ineffectual Code	Informational	TrustedPriceOracle.sol#L 43

[INFORMATIONAL] Description:

The specified line requires that either `now > timestamps[_latestId]` or `_latestId == 0` evaluates to `true`.

Recommendations:

In Solidity, the full address space of a `mapping` is initialized to its default value, in the case of a `uint256` being `0`, meaning that even if `_latestId` is zero, `now > timestamps[_latestId]` will evaluate to `true` since `now` is guaranteed to be a non-zero value as it represents a unix timestamp. As such, the latter part of the "OR" clause can be safely omitted.

Alleviation:

The latter part of the conditional was properly omitted from the relevant code.

TPO-04

TITLE	TYPE	SEVERITY	LOCATION
`require` to Modifier	Coding Style	Informational	TrustedPriceOracle#L66, L74 & L110

[INFORMATIONAL] Description:

The `require` statement in the aforementioned lines could instead be set to a contract `modifier` to ensure consistency in the way the check is carried out as currently, changes like the one mentioned in TPO-01 would require duplicate code changes.

Recommendations:

Included in the description above.

Alleviation:

Proper `modifier`s were defined for the linked `require` statements increasing the maintainability of the codebase.

MarketOracle.sol (MTO)

This contract is meant to coordinate with a primary and a fallback secondary oracle to answer oracle queries as well as calculate the volatility of the price of a particular asset. This is done based on the mathematical formulas defined in the Parameter Tuning White Paper.

MTO-01

TITLE	TYPE	SEVERITY	LOCATION
Redundant Explicit Assignment of Default Value	Ineffectual Code	Informational	MarketOracle.sol#L58

[INFORMATIONAL] Description:

The aforementioned line assigns the value `false` to the `bool` contract variable `isRecoveryPhase`.

Recommendations:

As Solidity assigns a default value to all declared variables without an assignment and the default value of a `bool` is false, this assignment is redundant.

Alleviation:

The assignment was properly omitted from the codebase.

MTO-02

TITLE	TYPE	SEVERITY	LOCATION
Function to Modifier Conversion	Coding Style	Informational	MarketOracle.sol#L180-L188

[INFORMATIONAL] Description:

The function `_recoveryPhaseCheck` is executed as the first statement of multiple functions and as such could be converted to a function `modifier` to aid in the legibility of the codebase.

Recommendations:

The current implementation and a `modifier` implementation are functionally equivalent.

Alleviation:

A `modifier` that carries out the exact same functionality was defined increasing the legibility of the codebase.

MTO-03

TITLE	TYPE	SEVERITY	LOCATION
Redundant Greater-Than-Or-Equal Comparison	Ineffectual Code	Informational	MarketOracle.sol#L97

[INFORMATIONAL] Description:

The SafeMath library's `sub` function internally conducts the exact same comparison carried out in the aforementioned line.

Recommendations:

As a result, it can be safely omitted.

If a custom error message is desired, it can be passed in as a second argument to the `sub` invocation as it is supported by the OpenZeppelin library.

It should be noted that the current order of `add` and `sub` should be reversed for this to work.

Alleviation:

The `require` message was omitted and a user-defined error message was passed in as the second argument to the `sub` invocation.

Fairswap Repository

The Integrated Fairswap repository is meant to implement a Uniswap-like exchange with additional protection mechanisms as well as a fee system that rewards LIEN token holders.

PriceCalculator.sol (PCR)

Implementation of functions to calculate trading price for a batch of orders.

PCR-01

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	PriceCalculator.sol#L36

[INFORMATIONAL] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons where applicable were properly converted to their inequality counterparts.

PCR-02

TITLE	TYPE	SEVERITY	LOCATION
Misleading notation	Coding Style	Informational	PriceCalculator.sol#L40, L17-L20, L67 & L76

[INFORMATIONAL] Description:

Consult captioned render of preliminary report below.

Recommendations:

On line 41 the first attempted price is defined as `(reserve1 + AmountSTRICT1_0 + AmountFLEX1_0) / (reserve0 + AmountSTRICT0_1 + AmountFLEX0_1)`. The corresponding mathematical formula in the second whitepaper page 18 is

$$P^* = \frac{X + \Delta X_M + \Delta X_L}{Y + \Delta Y_M + \Delta Y_L}$$

which implies `AmountSTRICT1_0, AmountFLEX1_0` represent buy orders and `AmountSTRICT0_1, AmountFLEX0_1` represent sell orders, in the comments lines 17-20 the definitions are swapped. Furthermore the value `AmountSTRICT1_0 + AmountFLEX1_0` is the input for `_calculateLowPrice` on line 67 as `allSellAmount` (line 99) and the value `AmountSTRICT0_1 + AmountFLEX0_1` is the input for `_calculateHighPrice` on line 76 as `allBuyAmount` (line 175), which does not follow the technical documentation. We recommend keeping the variable names consistent with the technical documentation.

Alleviation:

The variable names were synced with the surrounding documentation.

PCR-03

TITLE	TYPE	SEVERITY	LOCATION
Misleading notation	Coding Style	Informational	PriceCalculator.sol#L172

[INFORMATIONAL] Description:

Based on comments lines 17-20, limit orders are called strict and no-limit (market) orders are called flexible, but in the implementation of both functions `_calculateHighPrice` and `_calculateLowPrice` these two definitions are swapped, which leads to confusion.

Recommendations:

We advise that the definitions are properly synchronized with their comments.

Alleviation:

The comments and definitions were properly synced.

PCR-04

TITLE	TYPE	SEVERITY	LOCATION
Incorrect formula	Mathematical	Informational	PriceCalculator.sol#L44

[INFORMATIONAL] Description:

The second whitepaper page 18 defines the inequalities comparing the attempted price with the low price threshold and high price threshold is strict, whereas the implementation on line 52 uses nonstrict inequalities. However the effect of this deviation is negligible.

Recommendations:

We advise that strict inequalities are set.

Alleviation:

Strict inequalities were properly imposed in the relevant conditional.

PCR-05

TITLE	TYPE	SEVERITY	LOCATION
Redundant case	Ineffectual Code	Informational	PriceCalculator#L134-L151 & L211-L228

[INFORMATIONAL] Description:

The `if-else` construction in block 211-228 corresponds to the case work in second whitepaper page 19. Under the assumption that next high price is less than P^- (see PCR-07) we have

$$\frac{X + \Delta X'_M}{Y + \Delta Y_M + \Delta Y_L} < \frac{X + \Delta X_M}{Y + \Delta Y_M + \Delta Y_L}$$

which implies $\Delta X'_M < \Delta X_M$, so the condition on line 211 is always is always satisfied, so the else block 221-228 is redundant. Indeed that case can never happen because otherwise all buy market order would have to be cancelled because of the condition:

$$\frac{X}{Y + \Delta Y_M + \Delta Y_L} \geq (1 + T)P_0$$

but we also have

$$P_0 = \frac{X}{Y} \geq \frac{X}{Y + \Delta Y_M + \Delta Y_L}$$

so $P_0 \geq (1 + T)P_0$
which is a contradiction. Likewise the `if-else` construction in block 134-150 is also redundant.

Recommendations:

We advise these cases are omitted.

Alleviation:

These cases were properly omitted from the codebase.

PCR-06

TITLE	TYPE	SEVERITY	LOCATION
Incorrect formula	Mathematical	Minor	PriceCalculator.sol#L44-L46 & L121-L123

[Minor] Description:

The second whitepaper page 18 defines the low price threshold as $(1+t)^{-1}P_0$ and the next low price threshold as $(1+T)^{-1}P_0$, but the implementation on line 44 and 121 is $(1-t)P_0$ and $(1-T)P_0$, respectively. Even though for normal values of P_0, t, T the difference between these two formulas is insignificant, the error can get exacerbated by big values of P_0, t, T so we recommend fixing them.

Recommendations:

We advise that the proper formula is utilized.

Alleviation:

The proper formula was implemented and utilized in the corresponding functions.

PCR-07

TITLE	TYPE	SEVERITY	LOCATION
Incorrect formula	Mathematical	Major	PriceCalculator.sol#L124-L202

[Major] Description:

The second whitepaper page 19 compares the next high price $(1+T)P_0$ and next low price $(1+T)^{-1}P_0$ with P^-, but on lines 202, 124 they are compared with P^*. This can have negative effect on the traded volumes, especially in the case where $P^- > (1+T)^{-1}P_0 > P^*$. We recommend fixing the formula accordingly.

Recommendations:

We advise that the proper formula is utilized.

Alleviation:

The proper formula was implemented and utilized in the corresponding functions.

SpreadCalculator.sol (SCR)

Implementation of functions to calculate the spread for Liquid Bond Token.

SCR-01

TITLE	TYPE	SEVERITY	LOCATION
Redundant SafeCasts	Ineffectual Code	Informational	SpreadCalculator.sol#L62, L71, L103, L125, L131 & L133

[Informational] Description:

The variable `SPREAD_RATE` is a `constant` variable meaning that it cannot change throughout the lifetime of the contract.

Recommendations:

As such, it is possible to instantly cast it to a `uint64` instead of performing a safe `toUint64` cast since it is known that the variable will fit within the bit space of a `uint64`.

Alleviation:

The redundant SafeCasts were removed from the codebase.

SCR-02

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary `else` Block	Optimization	Informational	SpreadCalculator.sol#L101-L104

[Informational] Description:

As the aforementioned `else` block is the last code segment executable by the function and all preceding `if` clauses contain a `return` statement, it is possible to move the code within the `else` block outside of it at the end of the function.

As an additional optimization, the variable `spreadRate` could be declared outside the `if` blocks and assigned a default value of `SPREAD_RATE` that is subsequently adjusted within each `if` / `else` block and finally returned at the end via safecasting.

Recommendations:

Included in the description above.

Alleviation:

The code block was optimized to the greatest extent possible according to our recommendations.

SCR-03

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary `else` Block	Optimization	Informational	SpreadCalculator.sol#L132-L134

[Informational] Description:

The full code block of `calculateSpreadByAssetVolatility` can be refactored in a similar fashion to SCR-02 whereby a top-level variable is declared that is subsequently returned after being safely cast.

Recommendations:

Included in the description above.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

SCR-04

TITLE	TYPE	SEVERITY	LOCATION
Redundant SafeMath Calculation	Ineffectual Code	Informational	SpreadCalculator.sol#L161

[Informational] Description:

The aforementioned line performs a safe `mul` operation between a `constant` variable and the value literal `50`.

Recommendations:

As `constant` variables are replaced in-place within a contract's code by the compiler and it is known that this particular multiplication will not overflow, the statement `DECIMAL.mul(50)` would be replaced by the result of the multiplication rather than utilize the safe math counterpart it currently does and declare a redundant in-memory variable.

Alleviation:

The linked multiplication was replaced with its direct representation.

SCR-05

TITLE	TYPE	SEVERITY	LOCATION
Misleading comment	Coding Style	Informational	SpreadCalculator.sol#L17

[Informational] Description:

The comment on line 17 indicates that `MAX_RATIONAL_ORACLE_VALUE` is trillion, but it only has 5 zeros more than `DECIMAL`, which implies its value is just one hundred thousands.

Recommendations:

We advise that the comment is properly adjusted.

Alleviation:

The comment was properly adjusted.

SCR-06

TITLE	TYPE	SEVERITY	LOCATION
Inexistent Event Emittance	Ineffectual Code	Minor	SpreadCalculator.sol#L17

[Minor] Description:

The third `if-else` block of the `calculateCurrentSpread` function does not emit a `CalculateSpread` event whilst under all other circumstances it emits one.

Recommendations:

We advise that a proper event is emitted in this scenario.

Alleviation:

A proper event emittance was included in this circumstance.

RateMath.sol (RMH)

This library provides utility methods for carrying out calculations utilizing a rate point multiplier.

No findings were found in this library at this iteration.

OrderBox.sol (OBX)

This library is meant to provide utility methods for managing a box of multiple orders of different types.

OBX-01

TITLE	TYPE	SEVERITY	LOCATION
Struct Member Optimization	Optimization	Informational	OrderBox.sol#L8-L9

[Informational] Description:

The `OrderBox` struct ends with a `uint64` and a `uint32` variable, totalling at 96 bits.

Recommendations:

As the EVM optimally works with 256-bit numbers, it is possible to increase either variable accordingly to bring the total to 256-bits as this would reduce the cost of read / write operations on these variables as no padding would need to be conducted by the EVM.

Alleviation:

Both variable sizes were increased to fit a 256-bit slot equally at 128-bits each.

OBX-02

TITLE	TYPE	SEVERITY	LOCATION
Potential Numerical Overflows	Mathematical	Minor	OrderBox.sol#L41-L42

[Minor] Description:

The two statements linked above conduct an unsafe addition and assignment of the incoming amount of an order.

Recommendations:

While the amount of a single member overflowing is highly unlikely, the `totalInAmount` of an `OrderBook` could potentially overflow in an unsafely implemented token.

Although an overflow within those two statements would indicate a grander issue with the token implementation itself, we advise that the SafeMath library is utilized regardless to ensure consistency in the project's codebase and account for all types of edge cases.

Alleviation:

A SafeMath addition was properly set for those statements.

ExecutionStatus.sol (EXS)

This library provides utility methods for retrieving the refund rate of a particular `BoxExecutionStatus` struct as well as certain `struct` definitions to aid in tracking the execution status of a particular box.

EXS-01

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Data Type	Optimization	Informational	ExecutionStatus.sol#L16

[Informational] Description:

The EVM utilizes less gas to operate on data types that are 256-bit in size and the `BookExecutionStatus` struct currently utilizes two 256-bit slots regardless of the data type of `nextIndex`.

Recommendations:

We advise that it is instead declared as `uint256` to reduce the gas cost of operations on it.

Alleviation:

The variable was properly set to a 256-bit data size.

Enums.sol (ENS)

This file contains the various enums utilized throughout the system for differentiating the order pairs as well as utility methods for discerning what type an enum is in a legible way.

No findings were found in this library at this iteration.

BoxExchange.sol (BEX)

This contract defines how a typical exchange box should be structured to represent a Fairswap. It is an `abstract` implementation that is meant to be overridden by derivative contracts that want to implement specific pairs such as an Ethereum based pair or a token based one.

BEX-01

TITLE	TYPE	SEVERITY	LOCATION
Variable Visibility Specifiers Missing	Coding Style	Informational	BoxExchange.sol#L38, L40 & L47

[Informational] Description:

The aforementioned variables have been declared as `immutable` yet lack a visibility specifier.

Recommendations:

We advise that a visibility specifier is strictly defined for them.

Alleviation:

A proper visibility specifier was defined for each variable declaration.

BEX-02

TITLE	TYPE	SEVERITY	LOCATION
Inefficient `storage` Pointer Utilization	Optimization	Informational	BoxExchange.sol#L176-L189

[Informational] Description:

The specified code block assigns the result of the `orderBoxes[boxNumber]` lookup to a `storage` pointer and subsequently only accesses the `orderBooks` member of the struct.

Recommendations:

It is possible to instead assign the `orderBooks` member directly to a function-level variable declaration of `mapping(OrderType => OrderBook) storage` to avoid the extraneous struct member lookups.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

BEX-03

TITLE	TYPE	SEVERITY	LOCATION
Unnecessary `else` Block	Ineffectual Code	Informational	BoxExchange.sol#L196-L198 & L787-L789

[Informational] Description:

The default value of a `uint256` in Solidity is zero and as such, the redundant assignment as well as the whole `else` block that surrounds it can be safely omitted in the aforementioned code blocks.

Recommendations:

Included in the description above.

Alleviation:

The corresponding `else` blocks were properly omitted.

BEX-04

TITLE	TYPE	SEVERITY	LOCATION
Code Duplication	Ineffectual Code	Informational	BoxExchange.sol#L283-L291

[Informational] Description:

The aforementioned `if` and `else` clauses conduct the exact same statements while utilizing different variables in the process.

Recommendations:

As such, we advise that either a simple swap occurs within the first `if` clause and the statements of the `else` block are adjusted to be variable agnostic or that an `internal` function is created that carries out these calculations and returns the results.

Alleviation:

An `internal` function was properly defined that conducts these calculations called `_calculateAmounts`.

BEX-05

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	BoxExchange.sol#L441 & L927

[Informational] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were properly optimized.

BEX-06

TITLE	TYPE	SEVERITY	LOCATION
Duplicate Conditional Check	Ineffectual Code	Informational	BoxExchange.sol#L520-L522

[Informational] Description:

The `if` conditional within the `for` loop breaks if evaluated to `true` and is subsequently re-evaluated before explicitly returning a few variables.

Recommendations:

The `return` statement of L534 can instead be moved directly and replace the `break` statement to `return` within the `for` loop.

Alleviation:

A direct `return` within the `for` loop was defined.

BEX-07

TITLE	TYPE	SEVERITY	LOCATION
Code Duplication	Ineffectual Code	Informational	BoxExchange.sol#L599-L630

[Informational] Description:

In a similar fashion to BEX-04, these code blocks could instead be converted to `internal` functions that are subsequently invoked. This may also remove the need for the explicitly declared new execution context.

Recommendations:

Included in the description above.

Alleviation:

An `internal` function was defined called `_calculateNewReserveAndMarketFeePool` that carries out the duplicate statements.

BEX-08

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Less-Than Loop Conditional	Optimization	Informational	BoxExchange.sol#L665, L697 & L924

[Informational] Description:

These `for` loops are sequentially increasing by 1 and are guaranteed to always be less than or equal to their upper bound.

Recommendations:

As Solidity inequalities are more gas efficient than less-than comparisons, we advise these conditionals to be changed to different-than comparisons.

Alleviation:

The conditionals of the linked `for` loops were properly optimized.

BEX-09

TITLE	TYPE	SEVERITY	LOCATION
Variable Underflow	Mathematical	Minor	BoxExchange.sol#L772-L774

[Minor] Description:

The function `_currentOpenBoxId` will underflow when the contract has just been deployed. This can lead to undefined behavior if `_addOrder` is called with insufficient checks in the `_isCurrentOpenBoxExpired` derivative implementation.

Recommendations:

We advise that a SafeMath `sub` invocation is utilized instead or a manual `require` check to ensure that the contract has been properly initialized.

Alleviation:

The Lien team did not apply this Exhibit as the derivative `_isCurrentOpenBoxExpired` implementations are deemed to accommodate for this case.

ETHBoxExchange.sol (EBE)

An implementation of BoxExchange with one of the pairs being Ethereum.

EBE-01

TITLE	TYPE	SEVERITY	LOCATION
Variable Visibility Specifiers Missing	Coding Style	Informational	ETHBoxExchange.sol#L11, L13 & L14

[Informational] Description:

The aforementioned variables have been declared as `immutable` yet lack a visibility specifier.

Recommendations:

We advise that a visibility specifier is strictly defined for them.

Alleviation:

Proper visibility specifiers were set for the linked variables.

EBE-02

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	ETHBoxExchange.sol#L64, L89, L111, L206 & L209

[Informational] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

Proper inequality comparisons replaced the linked inefficient comparisons.

EBE-03

TITLE	TYPE	SEVERITY	LOCATION
`require` to Modifier	Coding Style	Informational	ETHBoxExchange.sol#L64-L65, L89-L90, L110-L111 & L135

[Informational] Description:

The `require` statement in the aforementioned lines could instead be set to a contract `modifier` to ensure consistency in the way the check is carried out as currently, changes like the one mentioned in EBE-02 would require duplicate code changes.

Recommendations:

Included in the description above.

Alleviation:

The `modifier`s were defined in the parent `BoxEchange.sol` contract.

EBE-04

TITLE	TYPE	SEVERITY	LOCATION
Code Duplication	Ineffectual Code	Informational	ETHBoxExchange.sol#L52-L98

[Informational] Description:

As the functions `orderEthToToken` and `orderTokenToEth` are quite similar, it is possible to instead utilize a common `internal` function. This, in combination with the `modifier` suggestions outlined in EBE-03 would greatly reduce code complexity and increase code legibility.

Recommendations:

Included in the description above.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

EBE-05

TITLE	TYPE	SEVERITY	LOCATION
Add Error Message to `require` Statement	Coding Style	Informational	ETHBoxExchange.sol#L185

[Informational] Description:

The aforementioned line conducts a `require` check without passing in an error message.

Recommendations:

While it is apparent what the statement is meant to conduct, it is a generally accepted coding practice to strictly set error messages for all types of `require` statements to aid in the debugging process.

Alleviation:

Proper error messages were added to the linked `require` calls.

IDOLvsETHBoxExchange.sol (IVE)

An implementation of ETHBoxExchange with the other pair being IDOL.

No findings were found in this library at this iteration.

ERC20vsETHBoxExchange.sol (EEE)

An implementation of ETHBoxExchange with the other pair being any ERC20 token.

EEE-01

TITLE	TYPE	SEVERITY	LOCATION
Code Duplication	Ineffectual Code	Informational	ERC20vsETHBoxExchange.sol#L49-L79

[Informational] Description:

These code blocks could instead be converted to `internal` functions that are subsequently invoked. This may also remove the need for the explicitly declared new execution context.

Recommendations:

Included in the description above.

Alleviation:

The `internal` function was defined in the parent `BoxExchange.sol` contract.

ERC20vsETHExchangeFactory.sol(EEF)

A factory for creating ERC20vsETHBoxExchange type Fairswaps.

EEF-01

TITLE	TYPE	SEVERITY	LOCATION
Variable Visibility Specifiers Missing	Coding Style	Informational	ERC20vsETHExchangeFactory.sol#L6, L7, L8 & L9

[Informational] Description:

The aforementioned variables have been declared as `immutable` yet lack a visibility specifier.

Additionally, the final variable is a state variable with no visibility specifier defined at all.

Recommendations:

We advise that a visibility specifier is strictly defined for them.

Alleviation:

Proper visibility specifiers were added for all linked variables.

EEF-02

TITLE	TYPE	SEVERITY	LOCATION
Misleading Variable Name	Coding Style	Informational	ERC20vsETHExchangeFactory.sol#L9

[Informational] Description:

The `mapping` variable `tokenToOracleToExchange` is actually used to map an oracle to a token and subsequently to an exchange.

Recommendations:

We advise that the variable name is updated to reflect that.

Alleviation:

The variable name was properly updated.

EEF-03

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	ERC20vsETHExchangeFactory.sol#L121

[Informational] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were properly converted to their inequality counterparts.

TokenBoxExchange.sol (TBE)

An implementation of BoxExchange with one of the pairs being iDOL.

TBE-01

TITLE	TYPE	SEVERITY	LOCATION
Variable Visibility Specifiers Missing	Coding Style	Informational	TokenBoxExchange.sol#L12, L13, L14 & L15

[Informational] Description:

The aforementioned variables have been declared as `immutable` yet lack a visibility specifier.

Recommendations:

We advise that a visibility specifier is strictly defined for them.

Alleviation:

Proper visibility specifiers were set for the linked variables.

TBE-02

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	TokenBoxExchange.sol#L74 & L99

[Informational] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were properly converted to their inequality counterparts.

TBE-03

TITLE	TYPE	SEVERITY	LOCATION
`require` to Modifier	Coding Style	Informational	TokenBoxExchange.sol#L74-L75, L99-L100, L122 & L159

[Informational] Description:

The `require` statement in the aforementioned lines could instead be set to a contract `modifier` to ensure consistency in the way the check is carried out as currently, changes like the one mentioned in TBE-02 would require duplicate code changes.

Recommendations:

Included in the description above.

Alleviation:

The `modifier`s were defined in the parent `BoxEchange.sol` contract apart for the latter two `require` statements which are duplicated.

TBE-04

TITLE	TYPE	SEVERITY	LOCATION
Code Duplication	Ineffectual Code	Informational	TokenBoxExchange.sol#L60-L108

[Informational] Description:

As the functions `orderBaseToSettlement` and `orderSettlementToBase` are quite similar, it is possible to instead utilize a common `internal` function. This, in combination with the `modifier` suggestions outlined in TBE-03 would greatly reduce code complexity and increase code legibility.

Recommendations:

Included in the description above.

Alleviation:

The Lien Protocol team took note of this Exhibit and chose not to apply it citing timeline constraints as it is merely an optimization.

LBTBoxExchange.sol (LBE)

An implementation of TokenBoxExchange for the pair of iDOL and LBT.

LBE-01

TITLE	TYPE	SEVERITY	LOCATION
Variable Mutability Type	Coding Style	Informational	LBTBoxExchange.sol#L7, L8 & L9

[Informational] Description:

The aforementioned variables have been declared as `public` yet are only assigned to once during the contract's `constructor`.

Recommendations:

We advise that the `immutable` mutability specifier is strictly defined for them.

Alleviation:

The proper mutability specifiers were set for the linked variables.

LBE-02

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	LBTBoxExchange.sol#L112

[Informational] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were properly converted to their inequality counterparts.

LBTExchangeFactory.sol(LEF)

A factory for creating LBTBoxExchange type Fairswaps.

LEF-01

TITLE	TYPE	SEVERITY	LOCATION
Variable Visibility Specifiers Missing	Coding Style	Informational	LBTExchangeFactory.sol# L8, L9, L10, L11, L12, L13, L14 & L15

[Informational] Description:

The aforementioned variables have been declared as `immutable` yet lack a visibility specifier.

Additionally, the final 2 variables are state variables with no visibility specifier defined at all.

Recommendations:

We advise that a visibility specifier is strictly defined for them.

Alleviation:

A proper visibility specifier was set for those variables.

LEF-02

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	LBTExchangeFactory.sol# L181

[Informational] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were properly converted to their inequality counterparts.

ERC20BoxExchange.sol(ERE)

An implementation of TokenBoxExchange for the pair of iDOL and any ERC20 token.

ERE-01

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	ERC20BoxExchange.sol# L46

[Informational] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were properly converted to their inequality counterparts.

ERE-02

TITLE	TYPE	SEVERITY	LOCATION
Code Duplication	Ineffectual Code	Informational	ERC20BoxExchange.sol# L68-L98

[Informational] Description:

These code blocks could instead be converted to `internal` functions that are subsequently invoked. This may also remove the need for the explicitly declared new execution context.

Recommendations:

Included in the description above.

Alleviation:

The `internal` function was defined in the parent `BoxExchange.sol` contract.

ERC20ExchangeFactory.sol (ERF)

A factory for creating ERC20BoxExchange type Fairswaps.

ERF-01

TITLE	TYPE	SEVERITY	LOCATION
Variable Visibility Specifiers Missing	Coding Style	Informational	ERC20ExchangeFactory.sol#L6, L7, L8, L9 & L10

[Informational] Description:

The aforementioned variables have been declared as `immutable` yet lack a visibility specifier.

Additionally, the final variable is a state variable with no visibility specifier defined at all.

Recommendations:

We advise that a visibility specifier is strictly defined for them.

Alleviation:

A proper visibility specifier was set for those variables.

ERC20Redistribution.sol (ERD)

A redistribution mechanism implementation for the LienBoxExchange to allow receipt of dividends by the exchange box.

ERD-01

TITLE	TYPE	SEVERITY	LOCATION
Inefficient Greater-Than Comparison w/ Zero	Optimization	Informational	ERC20Redistribution.sol# L32

[Informational] Description:

The comparison(s) being referred to by this Exhibit are being done so when the variables being compared will logically never be less-than zero due to f.e. their type being an unsigned integer such as `uint256` and thus being unable to go to the negative range.

Recommendations:

As inequality comparisons cost less gas than greater-than comparisons, it is optimal to convert the aforementioned comparison(s) to inequality comparison(s) with zero.

Alleviation:

The comparisons were properly converted to their inequality counterparts.

ERD-02

TITLE	TYPE	SEVERITY	LOCATION
Variable Mutability Type	Coding Style	Informational	ERC20Redistribution.sol# L17

[Informational] Description:

The aforementioned variable has been declared as `public` yet is only assigned to once during the contract's `constructor`.

Recommendations:

We advise that the `immutable` mutability specifier is strictly defined for them.

Alleviation:

The proper mutability specifiers were set for the linked variables.

ERD-03

[Informational] Description:

The contract in question essentially re-implements the `ERC20RegularlyRecord` contract from the `LienToken` repository.

Recommendations:

As such, we advise that the other repository is linked as a git submodule and that the corresponding source file is simply extended. Additionally, any Exhibits applicable to `ERC20RegularlyRecord` (ERR) are applicable to this contract as well.

Alleviation:

N/A

LienBoxExchange.sol (LIE)

An implementation of TokenBoxExchange for the pair of iDOL and LIEN.

No findings were found in this library at this iteration.

Round 2 Audit

A second round of audit commenced once the initial audit was complete that examined the codebase of the Lien Protocol under the viewpoint of edge case coverage and comprehensive unit testing.

During this process, we inspected and assimilated all existing tests of the contracts and proposed where necessary additional suites that should cover untested scenarios. This process led to bugs surfacing along with some additional minor optimizations that could be made to the project. The results of this process can be found below.

BondMaker.sol (BMK2)

This contract implementation provides utility methods for creating issuing new bonds, interacting with the Polyline implementation for price calculations as well as other conversion methods such as from bonds to ETH.

BMK2-01

TITLE	TYPE	SEVERITY	LOCATION
Locked Funds	Logical	Medium	BondMaker.sol#L340 - L409

[MEDIUM] Description:

It is possible to invoke the “exchangeEquivalentBonds” function regardless of whether the bonds have expired or not. If the bonds have expired but not been liquidated, a slippage of ETH occurs that will be forever locked in the contract.

Recommendations:

We advise that a check is imposed to prevent the execution of the “exchangeEquivalentBonds” function in case the bonds have expired.

Alleviation:

The Lien team added an additional check past the equal expiration check that ensures the current block time precedes the expiration time.

BMK2-02

TITLE	TYPE	SEVERITY	LOCATION
Trailing Funds	Mathematical	Minor	BondMaker.sol#L280

[MINOR] Description:

The linked division can possibly truncate trailing Ethereum digits, meaning that it is possible for funds albeit miniscule to remain locked / unaccounted for in the contract forever.

Recommendations:

We advise that a “require” check is imposed to ensure that the operation will not yield any trailing funds.

Alleviation:

The Lien team chose to instead provide extensive documentation as well as SDK-imposed limits to this function instead of a “require” check to save the gas increase as the trailing funds were found to be miniscule.

Auction.sol (AUC2)

The main implementation of a decentralized Vickrey auction whereby bids are secret. Contains all the relevant calculations for handling the full lifecycle of the auction.

AUC2-01

TITLE	TYPE	SEVERITY	LOCATION
Locked Auction	Logic	Medium	Auction.sol#L423 - L425

[MEDIUM] Description:

Due to the way “makeAuctionResult” functions, it was possible for an auction to make the function unexecutable under the circumstance that the final price became higher than the strike price as bids are allowed to exceed the strike price by 10% as the SafeMath sub invocation would throw.

Recommendations:

We advise that instead of conducting a SafeMath invocation, a manual “if” clause is imposed whereby the variable “toPay” is checked and, in case it exceeds “auctionLockedIDOLAmountE8”, it is set to its value.

Alleviation:

The Lien team applied our recommendation on the codebase to the letter.

AUC2-02

TITLE	TYPE	SEVERITY	LOCATION
Incorrect “require” Message	Textual	Informational	Auction.sol#L179

[INFORMATIONAL] Description:

The linked “require” check contains an incorrect error message whereby it states that the “loser” bids are not sorted whilst the function concerns “winning” bids.

Recommendations:

We advise that the error message reflects the context of the function.

Alleviation:

The error message was properly set to reflect the “winning” bids not being sorted.

AUC2-03

TITLE	TYPE	SEVERITY	LOCATION
Code Duplication	Ineffectual Code	Informational	Auction.sol#L635 - L640

[INFORMATIONAL] Description:

The linked code block replicates the exact same statements as the “_distributeToWinners” function.

Recommendations:

We advise that the function is invoked instead to reduce the bytecode of the contract.

Alleviation:

The code block was replaced by the function invocation properly.

DecentralizedOTC.sol (OTC2)

The decentralized over-the-counter market implementation of Lien that enables participants to create pools of ERC20 tokens with their respective price calculator oracles. Contains logic that calculates the exchange rates between an LBT and ERC20 pair as well as configuration methods to properly set up the corresponding pools.

OTC2- 01

TITLE	TYPE	SEVERITY	LOCATION
Usage of “abi.encodePacked” for Identifiers	Coding Style	Informational	DecentralizedOTC.sol#L90 - L96

[INFORMATIONAL] Description:

The function “abi.encodePacked” is utilized within DecentralizedOTC to derive the ID of an ERC20 pool. The variables utilized are actually tight-packed by the ABI encoder, meaning that a potential hash collision could occur depending on the variables.

Recommendations:

Although it may be practically infeasible in the current implementation, we advise that the “abi.encode” function is utilized instead to ensure that a malicious collision based on input can occur under no circumstance.

Alleviation:

The Lien team properly changed the encoding mechanism to the bare one.

StableCoin.sol (STC2)

The iDOL derivative backed stablecoin implementation that interacts with derivative auctions. Contains core logic of a token as well as the corresponding auction-based minting mechanisms w/ solid bond tokens.

STC2-01

TITLE	TYPE	SEVERITY	LOCATION
Invalid Log Message	Race Condition	Informational	StableCoin.sol#L716

[INFORMATIONAL] Description:

The “returnLockedPoolTo” function contains no strict access control meaning that it can be invoked by anyone for any address.

This can in turn lead to race conditions occurring on blockchain whereby someone scans the mempool of an Ethereum node and invokes the function on behalf of an existing transaction to cause invalid log messages and application behavior on the end of the original caller.

Recommendations:

We advise the usefulness of being callable by anyone for this particular function is carefully assessed and if necessary, log messages are properly emitted.

Alleviation:

An additional check was imposed prior to emitting the “LogReturnLockedPool” event to ensure that a non-zero value was returned.

AuctionBoard.sol (AUB2)

Contains the core logic for the lifecycle of bids in a particular auction, including all types of utility methods such as sorting the bids of an auction.

AUB2-01

TITLE	TYPE	SEVERITY	LOCATION
Redundant “require” Constructor Statements	Coding Style	Informational	AuctionBoard.sol#L149 - L160

[INFORMATIONAL] Description:

The linked “require” statements ensure that the input variables of the constructor are within the range of a specific data type less than the variable data type.

Recommendations:

We advise that the input variables of the constructor are instead set to their supposed data type to be able to omit the “require” checks as they would be guaranteed.

Alleviation:

Our recommendation was followed to the letter.

Helper.sol (HLP)

A new contract that was introduced in an update and provides utility methods for either front-end operations or general contract based functionality.

HLP-01

TITLE	TYPE	SEVERITY	LOCATION
Visibility Specifiers Missing	Coding Style	Informational	Helper.sol#L21

[INFORMATIONAL] Description:

The aforementioned variables lack a visibility specifier.

Recommendations:

We advise that their visibility is properly set.

Alleviation:

Their visibility specifiers were strictly set nullifying this Exhibit.

HLP-02

TITLE	TYPE	SEVERITY	LOCATION
“revert” to “require” Clause	Coding Style	Informational	Helper.sol#L57 - L59

[INFORMATIONAL] Description:

The linked “else” block contains a “revert” statement with an error message.

Recommendations:

As “revert” statements are ill-advised in Solidity, we instead suggest that a “require” check is imposed preceding the “if” block that guarantees the last valid condition of the “if-else” chain.

Alleviation:

N/A.

LienPriceOracle.sol (LPO)

A new contract that was introduced in an update and provides the price of Lien as a utility contract.

LPO-01

TITLE	TYPE	SEVERITY	LOCATION
Centralized Lower Bound	Logical	Informational	LienPriceOracle.sol#L71 - L73

[INFORMATIONAL] Description:

The lower bound of the reported price is set via an administrative function of the contract, meaning that if the oracle reported price is lower than the bound, the bound will be returned instead.

Recommendations:

Although we understand this is a side-effect of what the contract is meant to achieve, we wanted to note that this brings high centralization to the price reporting mechanism of the involved contract.

Alleviation:

The Lien team responded by saying this limit was imposed as a matter of provider preference rather than centralization, meaning that multiple oracles each with its own respective lower bound provide a decentralized price report.

Final Round Conclusion

The Lien team engaged in real-time communication with our auditing team throughout the whole auditing process and was very responsive with regards to any update they made in the codebase that could potentially affect our process and / or the findings we had provided.

During the audit process, the Lien team conducted extensive tests and aggressively expanded their unit tests to ensure a wide range of scenarios are covered for their contracts and that they behave as expected even when put under fuzzing tests. By utilizing fuzzing, they ensure that discretized random numbers are applied to the codebase instead that have a much higher probability of triggering edge cases.

Overall, the Lien team has showcased a very conservative approach to their product by acquiring a Consensys Dilligence audit, deploying an alpha version on the Ropsten network for livenet testing for several weeks, procure our auditing services (CertiK) and in parallel deploy the contracts on the Mainnet integrated with the application front-ends to optimize event emittences, input limits and other types of issues that would have arisen to this point.

Due to the aforementioned approach, it is safe to assume that the Lien project is of very high quality with a good security score and a cross-audited codebase.