CERTIK

Lien Protocol

Smart Contracts

Security Assessment

January 20th, 2021

By:
Adrian Hetman @ CertiK
adrian.hetman@certik.org

Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

What is a CertiK report?

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

Overview

Project Summary

Project Name	**Lien Protocol**
Description	A protocol for creating Options and Stablecoins out of ETH supporting curve-based bond generation formulas and oracle-based native multi-type bond exchanges.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. 176af5f323d4c291cc8cf6a512c874b7752fc733 2. f73cbb64b8475693cc4937679c37c58410a1cbf9 3. ca361bb664b5ced9e2cdde119d55e92c0670a0f2

Audit Summary

Delivery Date	**January 20th, 2021**
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	3
Timeline	November 23rd, 2020 - December 11th, 2020

Vulnerability Summary

Total Issues	**32**
Total Critical	0
Total Major	7
Total Medium	6
Total Minor	9
Total Informational	10

Executive Summary

This report has been prepared for Lien to discover issues and vulnerabilities in the source code of their Bond Token, Generalized Over the Counter exchange, and Bond options Smart Contracts. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The codebase had few issues which the team has addresses, mostly regarding address verification and access control on bond exchange. Overall code is of good quality, following best practices. It's worth noting that apart from a few major and medium issues, no critical vulnerabilities were found, and the team was swift and keen to promptly fix the issues.

We feel Bond option functionality is working as intended, based on the documentation and our conversation with the team.Our biggest concern was around flash-loan and oracle based attacks, but we haven't found any indications of such attacks being possible. We feel confident in saying no issues were detected on that front, especially on Chainlink Oracles; looking at the current landscape of attacks and new possibilities of doing batch-flash-loans on Aave V2; new attacks could emerge. Lien has fixed most of the issues, but few exhibits were acknowledged but not address, such as lack of address verification that `BondPricer` is a valid contract.

The project's mathematical side was also checked, and few issues were found, but nothing of the critical matter and the team addressed them all. One point still needs to be worked on in Polyline Smart Contract regarding redundant functions as it would require extensive modification to the codebase. Still, the team has acknowledged the issue and will be working on it in the future.

The team merged audit branch with the public one and commit for audit branch has changed to commit number 3 in our commit summary.

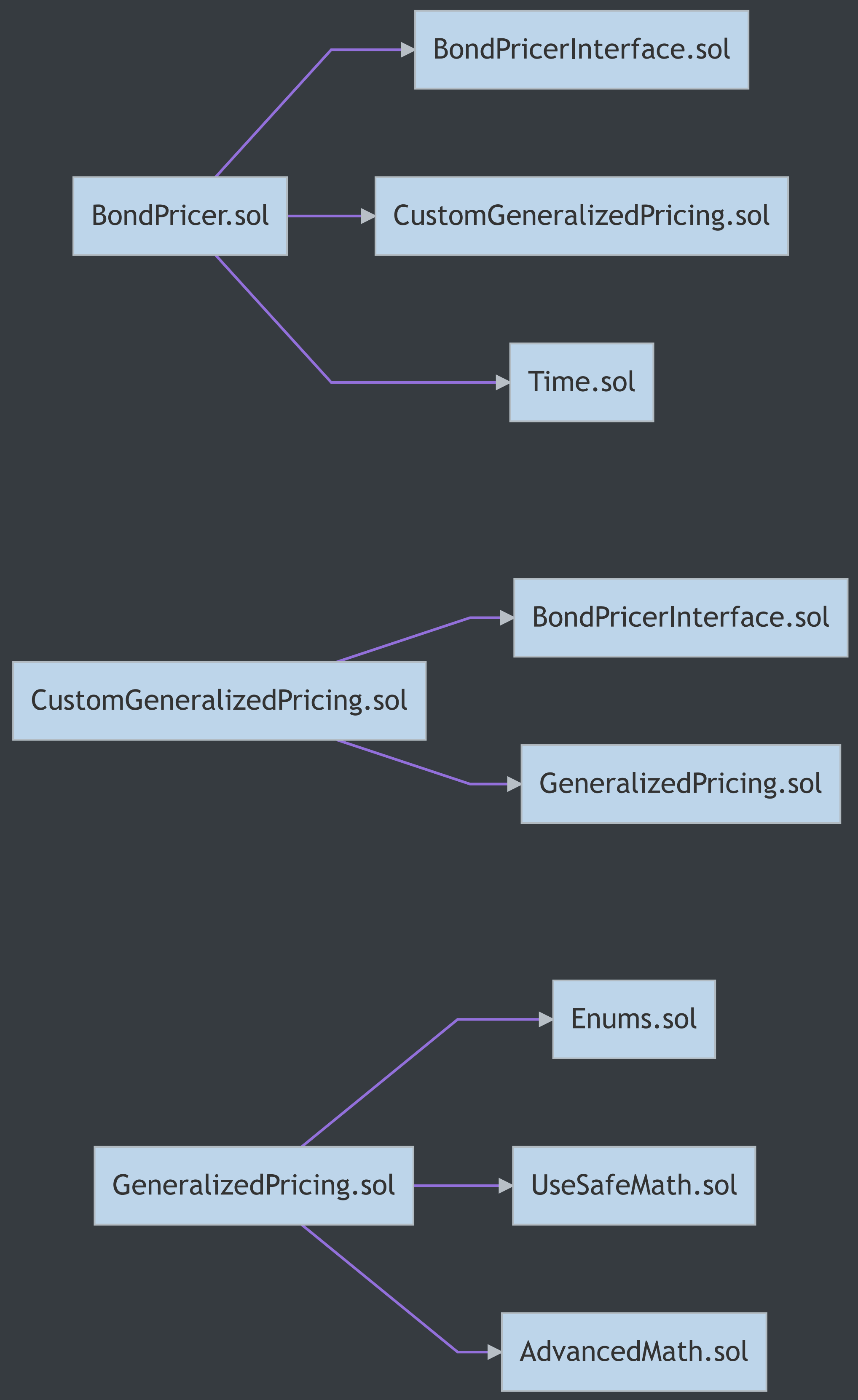
Files In Scope

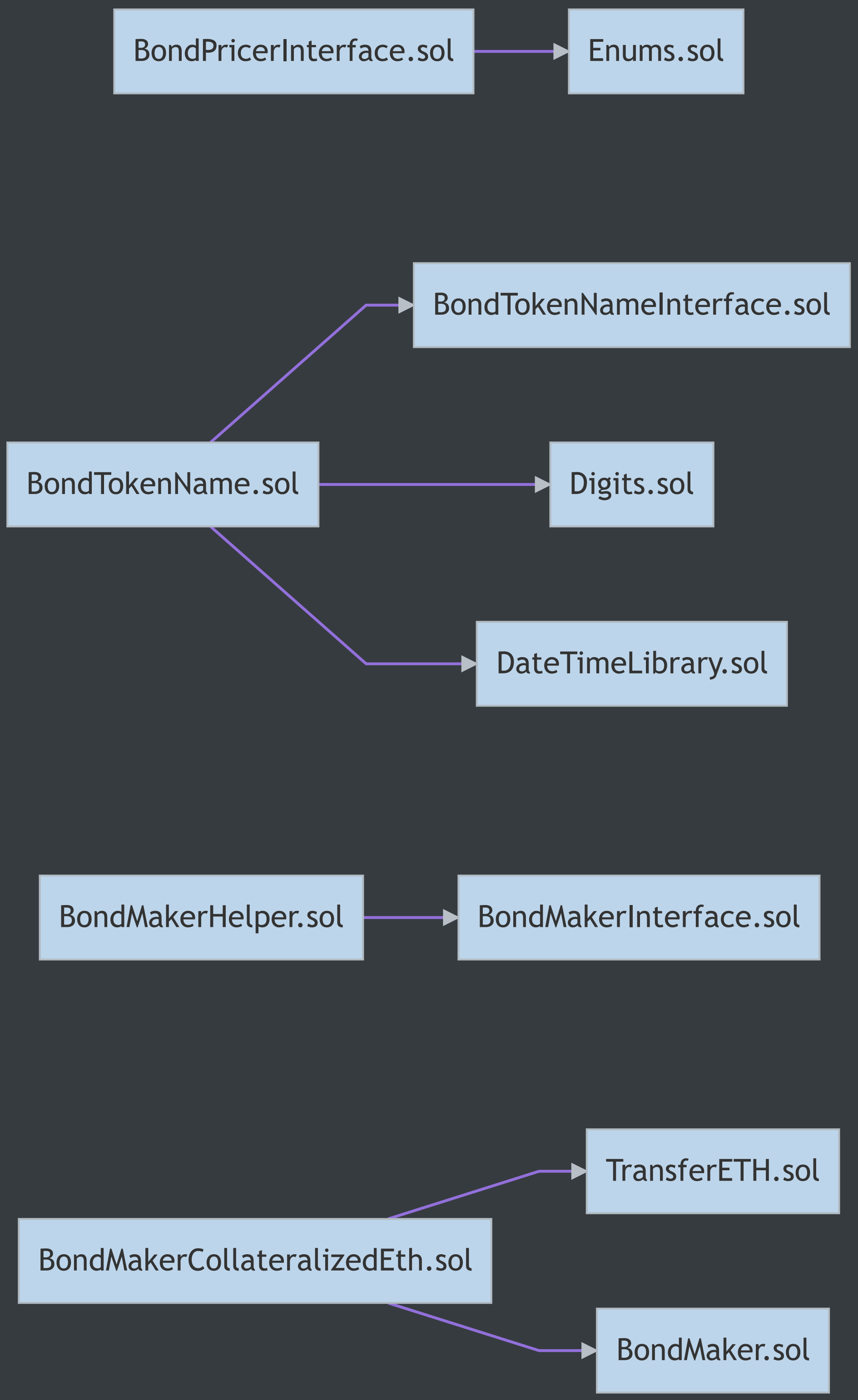
ID	Contract	Location
AMH	AdvancedMath.sol	contracts/math/AdvancedMath.sol
BMR	BondMaker.sol	contracts/bondMaker/BondMaker.sol
BTN	BondToken.sol	contracts/bondToken/BondToken.sol
BPR	BondPricer.sol	contracts/bondPricer/BondPricer.sol

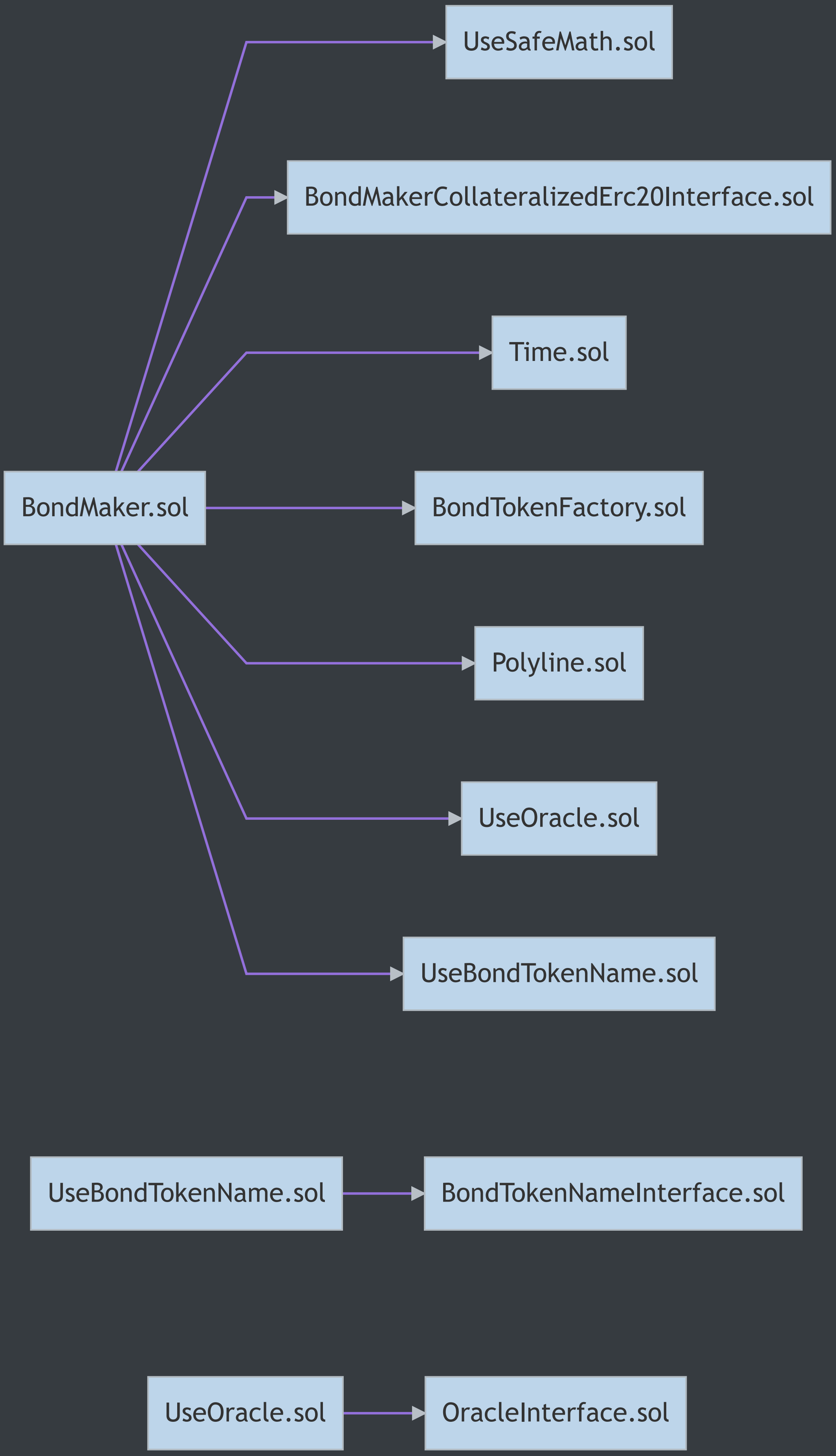
BEE	BondExchange.sol	contracts/generalizedDotc/BondExchange.sol
CON	BondTokenName.sol	contracts/bondTokenName/BondTokenName.sol
BMH	BondMakerHelper.sol	contracts/helper/BondMakerHelper.sol
BTF	BondTokenFactory.sol	contracts/bondToken/BondTokenFactory.sol
BVE	BondVsEthExchange.sol	contracts/generalizedDotc/BondVsEthExchange.sol
BMI	BondMakerInterface.sol	contracts/bondMaker/BondMakerInterface.sol
BTI	BondTokenInterface.sol	contracts/bondToken/BondTokenInterface.sol
BVB	BondVsBondExchange.sol	contracts/generalizedDotc/BondVsBondExchange.sol
BPI	BondPricerInterface.sol	contracts/bondPricer/BondPricerInterface.sol
VEE	BondVsErc20Exchange.sol	contracts/generalizedDotc/BondVsErc20Exchange.sol
BNI	BondTokenNameInterface.sol	contracts/bondTokenName/BondTokenNameInterface.sol
BMC	BondMakerCollateralizedEth.sol	contracts/bondMaker/BondMakerCollateralizedEth.sol
BTC	BondTokenCollateralizedEth.sol	contracts/bondToken/BondTokenCollateralizedEth.sol
BME	BondMakerCollateralizedErc20.sol	contracts/bondMaker/BondMakerCollateralizedErc20.sol
BTE	BondTokenCollateralizedErc20.sol	contracts/bondToken/BondTokenCollateralizedErc20.sol
BCI	BondMakerCollateralizedEthInterface.sol	contracts/bondMaker/BondMakerCollateralizedEthInterface.sol
BEI	BondMakerCollateralizedErc20Interface.sol	contracts/bondMaker/BondMakerCollateralizedErc20Interface.sol
CGP	CustomGeneralizedPricing.sol	contracts/bondPricer/CustomGeneralizedPricing.sol
DIG	Digits.sol	contracts/util/Digits.sol
DTL	DateTimeLibrary.sol	contracts/util/DateTimeLibrary.sol
DBS	DetectBondShape.sol	contracts/bondPricer/DetectBondShape.sol
ENU	Enums.sol	contracts/bondPricer/Enums.sol
ERM	ERC20Mintable.sol	contracts/token/ERC20Mintable.sol
ERV	ERC20Vestable.sol	contracts/token/ERC20Vestable.sol
EBM	EthBondMakerCollateralizedUsdc.sol	contracts/bondMaker/EthBondMakerCollateralizedUsdc.sol
FPO	FixedPriceOracle.sol	contracts/oracle/FixedPriceOracle.sol
FSP	FairSwapPriceOracle.sol	contracts/oracle/FairSwapPriceOracle.sol
GDC	GeneralizedDotc.sol	contracts/generalizedDotc/GeneralizedDotc.sol
GPG	GeneralizedPricing.sol	contracts/bondPricer/GeneralizedPricing.sol

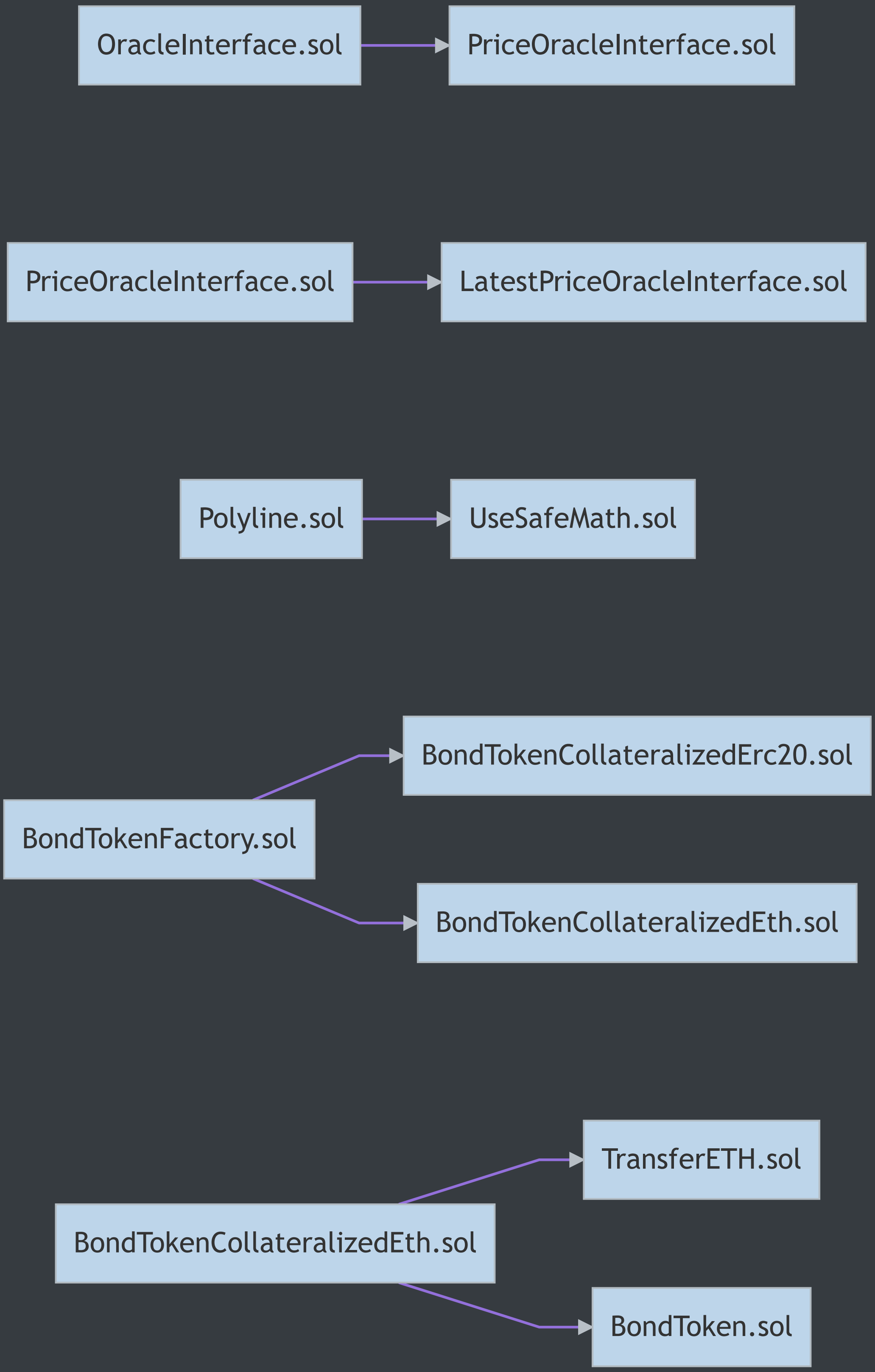
LPO	LienPriceOracle.sol	contracts/oracle/LienPriceOracle.sol
LPI	LatestPriceOracleInterface.sol	contracts/oracle/LatestPriceOracleInterface.sol
OIE	OracleInterface.sol	contracts/oracle/OracleInterface.sol
POL	Polyline.sol	contracts/util/Polyline.sol
PIO	PriceInverseOracle.sol	contracts/oracle/PriceInverseOracle.sol
POI	PriceOracleInterface.sol	contracts/oracle/PriceOracleInterface.sol
SPW	SbtPricerWithStableBorder.sol	contracts/bondPricer/SbtPricerWithStableBorder.sol
TIM	Time.sol	contracts/util/Time.sol
TET	TransferETH.sol	contracts/util/TransferETH.sol
TEH	TransferETHInterface.sol	contracts/util/TransferETHInterface.sol
UOE	UseOracle.sol	contracts/oracle/UseOracle.sol
USC	USDCOracle.sol	contracts/oracle/USDCOracle.sol
USM	UseSafeMath.sol	contracts/math/UseSafeMath.sol
UBM	UseBondMaker.sol	contracts/bondMaker/UseBondMaker.sol
UBT	UseBondTokenName.sol	contracts/bondTokenName/UseBondTokenName.sol
UBC	UseBondMakerCollateralizedEth.sol	contracts/bondMaker/UseBondMakerCollateralizedEth.sol

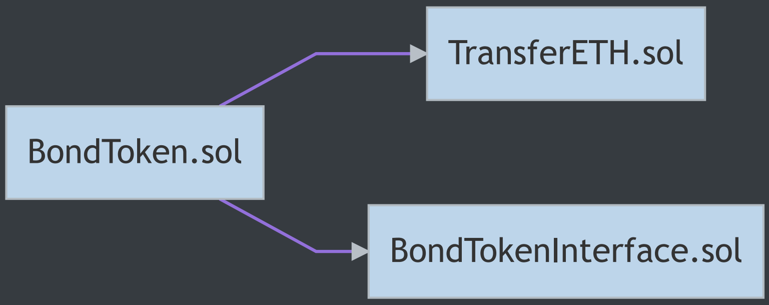
File Dependency Graph

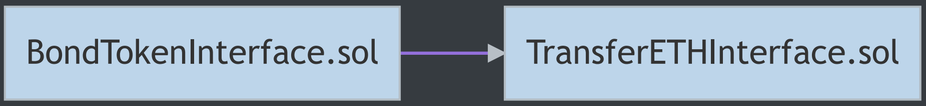


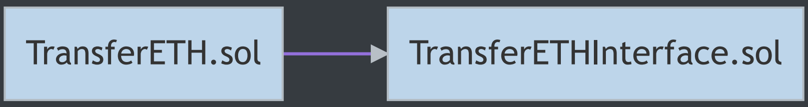

BondPricerInterface.sol
Enums.sol
BondTokenNameInterface.sol
BondTokenName.sol
Digits.sol
DateTimeLibrary.sol
BondMakerHelper.sol
BondMakerInterface.sol
TransferETH.sol
BondMakerCollateralizedEth.sol
BondMaker.sol

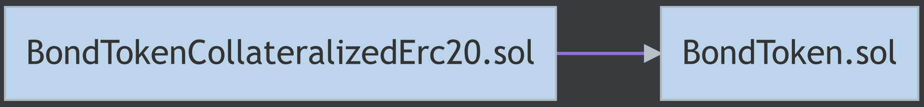

BondMaker.sol
UseSafeMath.sol
BondMakerCollateralizedErc20Interface.sol
Time.sol
BondTokenFactory.sol
Polyline.sol
UseOracle.sol
UseBondTokenName.sol
UseBondTokenName.sol
BondTokenNameInterface.sol
UseOracle.sol
OracleInterface.sol

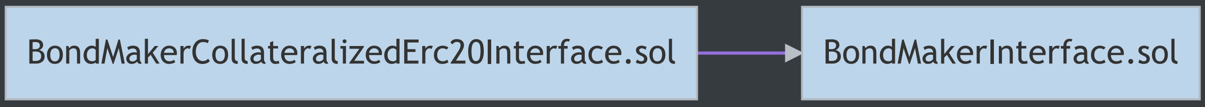

OracleInterface.sol
PriceOracleInterface.sol
PriceOracleInterface.sol
LatestPriceOracleInterface.sol
Polyline.sol
UseSafeMath.sol
BondTokenFactory.sol
BondTokenCollateralizedErc20.sol
BondTokenCollateralizedEth.sol
BondTokenCollateralizedEth.sol
TransferETH.sol
BondToken.sol

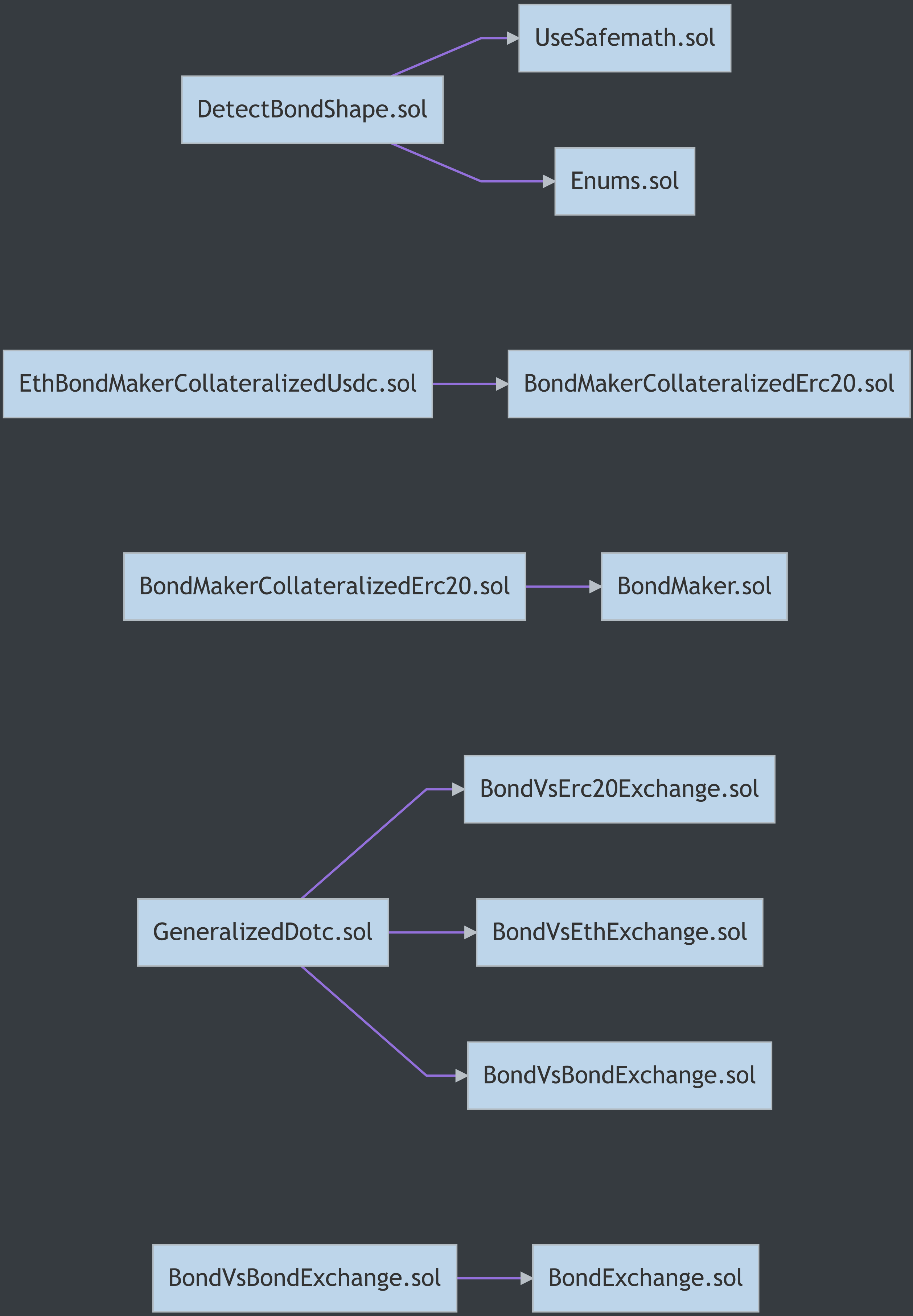

BondToken.sol
TransferETH.sol
BondTokenInterface.sol

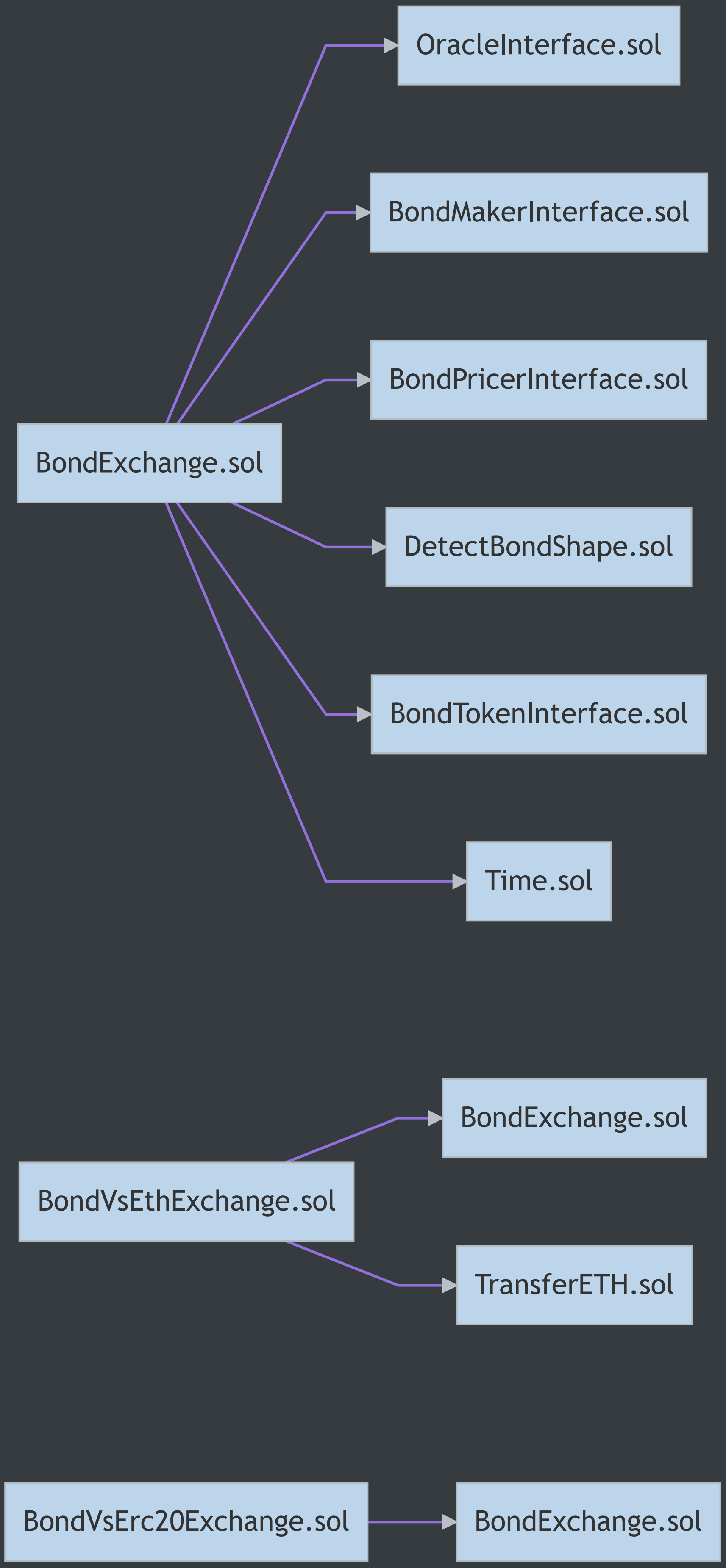

BondTokenInterface.sol
TransferETHInterface.sol

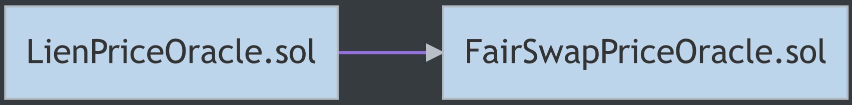

TransferETH.sol
TransferETHInterface.sol

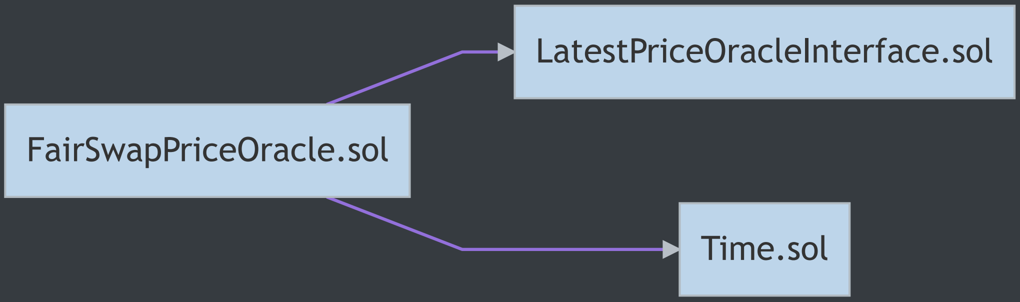

BondTokenCollateralizedErc20.sol
BondToken.sol

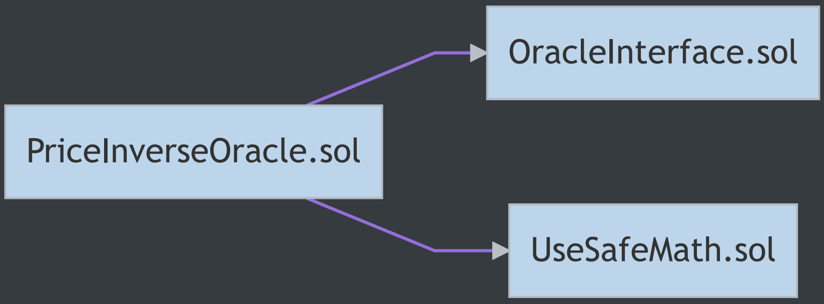

BondMakerCollateralizedErc20Interface.sol
BondMakerInterface.sol

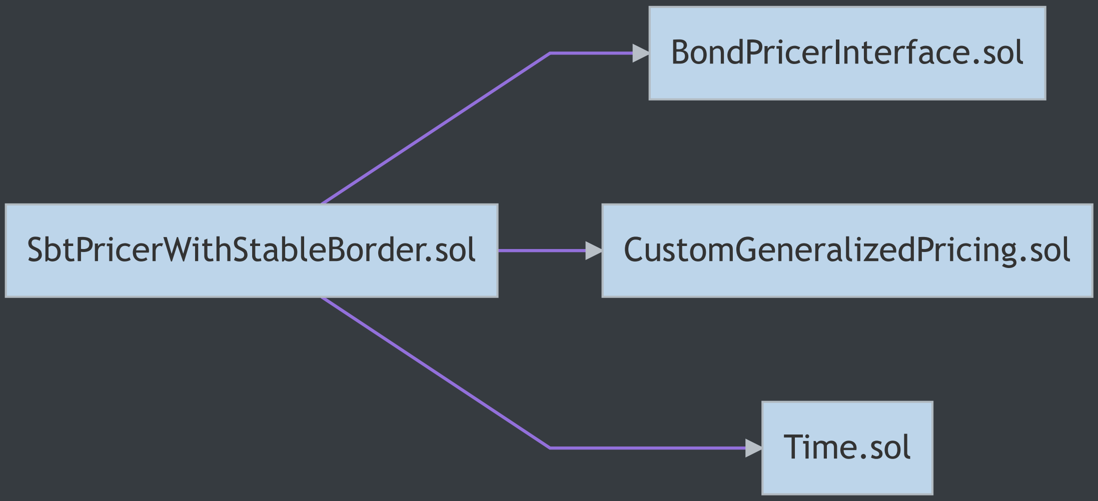

DetectBondShape.sol
UseSafemath.sol
Enums.sol
EthBondMakerCollateralizedUsdc.sol
BondMakerCollateralizedErc20.sol
BondMakerCollateralizedErc20.sol
BondMaker.sol
GeneralizedDotc.sol
BondVsErc20Exchange.sol
BondVsEthExchange.sol
BondVsBondExchange.sol
BondVsBondExchange.sol
BondExchange.sol

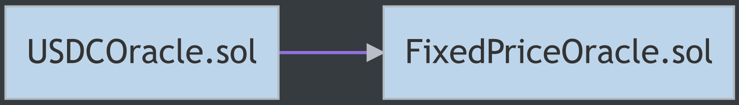

BondExchange.sol
OracleInterface.sol
BondMakerInterface.sol
BondPricerInterface.sol
DetectBondShape.sol
BondTokenInterface.sol
Time.sol
BondVsEthExchange.sol
BondExchange.sol
TransferETH.sol
BondVsErc20Exchange.sol
BondExchange.sol

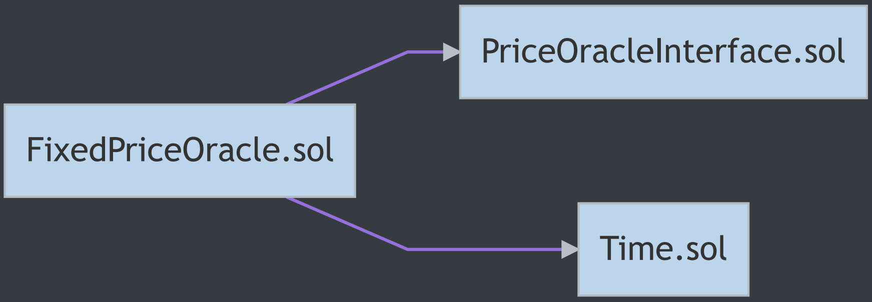

LienPriceOracle.sol
FairSwapPriceOracle.sol

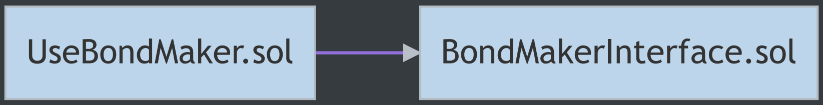

FairSwapPriceOracle.sol
LatestPriceOracleInterface.sol
Time.sol

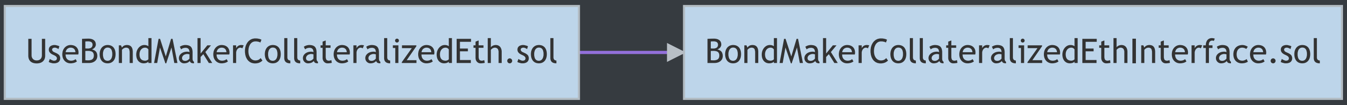

PriceInverseOracle.sol
OracleInterface.sol
UseSafeMath.sol

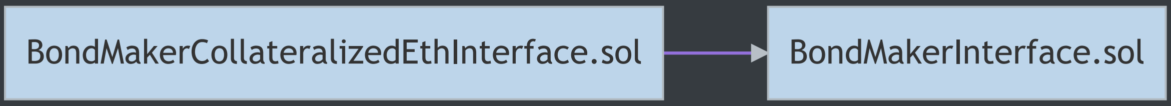

SbtPricerWithStableBorder.sol
BondPricerInterface.sol
CustomGeneralizedPricing.sol
Time.sol


USDCOracle.sol
FixedPriceOracle.sol

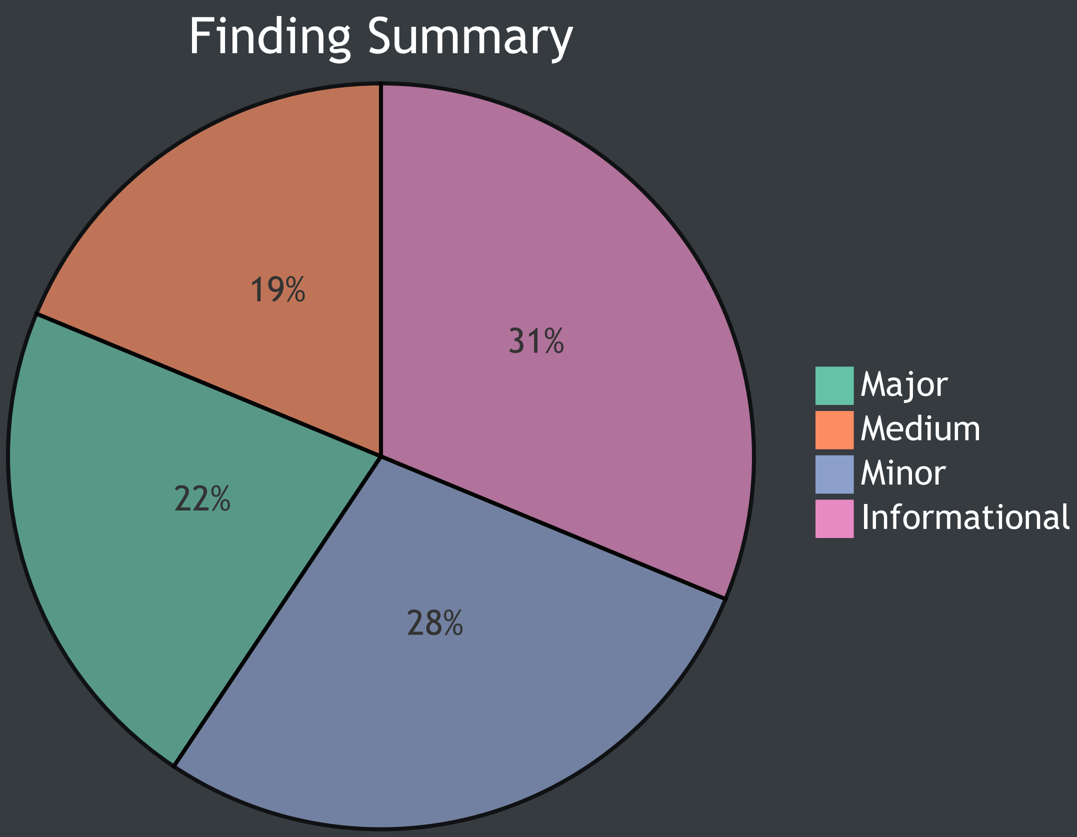

FixedPriceOracle.sol
PriceOracleInterface.sol
Time.sol


UseBondMaker.sol
BondMakerInterface.sol


UseBondMakerCollateralizedEth.sol
BondMakerCollateralizedEthInterface.sol


BondMakerCollateralizedEthInterface.sol
BondMakerInterface.sol

Findings



ID	Title	Type	Severity	Resolved
BME-01	Unchecked Value of ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	✓
BME-02	`issueNewBonds` for ERC20 tokens doesn't take into account potential fees for ERC20.	Volatile Code	Minor	✓
BTE-01	Unchecked Value of ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	✓
CON-01	Potential collision on names	Volatile Code	Minor	!
CON-02	Unlocked Compiler Version	Language Specific	Informational	✓
BEE-	Argument name and function name are misleading	Inconsistency	Informational	

01				✓
BVB-01	Lack of restrictions for `deleteVsBondPool`	Control Flow	Major	✓
BVB-02	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	⏱
BVB-03	Lack of address verification	Volatile Code	Medium	✓
BVB-04	Lack of address verification for passed `bondPricerAddress` .	Volatile Code	Major	⏱
VEE-01	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	⏱
VEE-02	Lack of address verification	Volatile Code	Medium	✓
VEE-03	Lack of address verification for passed `bondPricerAddress` .	Volatile Code	Major	⏱
VEE-04	Lack of restrictions for `deleteVsErc20Pool`	Control Flow	Major	✓
BVE-01	Lack of address verification	Volatile Code	Medium	✓
BVE-02	Lack of address verification for passed `bondPricerAddress` .	Volatile Code	Major	⏱
BVE-03	Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call	Logical Issue	Minor	⏱
BVE-04	Lack of restrictions for `deleteVsEthPool`	Control Flow	Major	✓
DTL-01	Divide before multiply	Mathematical Operations	Minor	⏱
DTL-02	Unlocked Compiler Version	Language Specific	Informational	✓
DIG-01	Unlocked Compiler Version	Language Specific	Informational	✓

BNI-01	Unlocked Compiler Version	Language Specific	Informational	✓
OIE-01	Unlocked Compiler Version	Language Specific	Informational	✓
BMR-01	Incompatible Structure	Logical Issue	Major	✓
BMR-02	Missing Requirement	Logical Issue	Minor	⏱
BMR-03	Inefficient Search	Gas Optimization	Informational	⏱
POL-01	Redundant Statements	Dead Code	Informational	⏱
POL-02	Potentially Redundant Functions	Gas Optimization	Medium	⏱
GPG-01	Inefficient Variable Type	Gas Optimization	Informational	⏱
GPG-02	Integer Overflow / Underflow	Mathematical Operations	Medium	✓
GPG-03	Incorrect Comment for `require`	Logical Issue	Informational	✓
DBS-01	Missing Requirement	Logical Issue	Medium	✓

BME-01: Unchecked Value of ERC-20 `transfer()` / `transferFrom()` Call

Type	Severity	Location
Volatile Code	Minor	BondMakerCollateralizedErc20.sol L32, L84

Description:

The linked `transfer()` / `transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation:

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation:

Issue Resolved

BME-02: `issueNewBonds` for ERC20 tokens doesn't take into account potential fees for ERC20.

Type	Severity	Location
Volatile Code	Minor	BondMakerCollateralizedErc20.sol L30-L34

Description:

Some ERC20 tokens include within their protocol fees which translate to lower actual amount received vs amount sent. This could lead to misscalculating allowance and expecting wrong amount of token to receive.

Recommendation:

Instead of passing `allowance` to `_issueNewBonds` we recommend saving balance before `transferFrom` and balance after, then passing subtraction of these two which would give actual amount received.

Alleviation:

Issue resolved.

BTE-01: Unchecked Value of ERC-20 `transfer()` / `transferFrom()` Call

Type	Severity	Location
Volatile Code	Minor	BondTokenCollateralizedErc20.sol L23

Description:

The linked `transfer()` / `transferFrom()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation:

As many tokens do not follow the ERC-20 standard faithfully, they may not return a `bool` variable in this function's execution meaning that simply expecting it can cause incompatibility with these types of tokens. Instead, we advise that OpenZeppelin's `SafeERC20.sol` implementation is utilized for interacting with the `transfer()` and `transferFrom()` functions of ERC-20 tokens. The OZ implementation optionally checks for a return value rendering compatible with all ERC-20 token implementations.

Alleviation:

Issue Resolved

CON-01: Potential collision on names

Type	Severity	Location
Volatile Code	Minor	BondTokenName.sol L51-L57

Description:

Current implementation for `getBondTokenName` for SBT, LBT and IMT could cause a colision for bond token that have same ticker, maturity and strike price.

Recommendation:

The Bond Token name generation should use additional contextual variables for generating a unique name even if the aforementioned variables are equal.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase.

CON-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	BondTokenName.sol L1

Description:

The contract has unlocked compiler version. 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.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue Resolved

BEE-01: Argument name and function name are misleading

Type	Severity	Location
Inconsistency	Informational	BondExchange.sol L175

Description:

`_calcVolumeOnUsdBasis` function usage across the various types of exchanges does not seem to be of calculating any volume (i.e. trading volume) but rather it's used as a price calculation function in unclear ways as its argument, named volume, is actually a price offset to E8.

Recommendation:

Rename function or argument to reflect actual logic.

Alleviation:

Issue Resolved

BVB-01: Lack of restrictions for `deleteVsBondPool`

Type	Severity	Location
Control Flow	Major	BondVsBondExchange.sol L130-L132

Description:

Anyone knowing a pool's ID could call `deleteVsBondPool` function and delete it as there is no ACL imposed.

Recommendation:

An ACL check should be imposed that verifies the caller of the deletion operation is the bond pool owner.

Alleviation:

Issue Resolved

BVB-02: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	BondVsBondExchange.sol L240, L412, L415

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true` , many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase to the Bond Token.

Client Comment:
"We didn’t fix this point for BondToken transfer because error message will be more confusing if use safeTransfer."

BVB-03: Lack of address verification

Type	Severity	Location
Volatile Code	Medium	BondVsBondExchange.sol L141-L144, L312-L314, L332-L333

Description:

The linked function implementations do not properly sanitize the `address` variables they utilize.

Recommendation:

Additional sanitization should be imposed to ensure proper operation of the system i.e. inequality checks with the zero address.

Alleviation:

Issue Resolved but require comment in L374 isn't correct.

BVB-04: Lack of address verification for passed `bondPricerAddress`.

Type	Severity	Location
Volatile Code	Major	BondVsBondExchange.sol L314, L333

Description:

There are no direct checks that the underlying code of the Bond Pricer contract is valid for a particular input Bond Pricer address.

Recommendation:

We recommend that the Bond Pricers that should be acceptable by the Exchange are generated via a factory that Lien provides, thus allowing an easy validation to occur by ensuring the contract was deployed via the factory and consequently has unaltered bytecode.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase.

Client Comment:
"Users will use the pool managed by Lien team (pools created by our address or Aggregator contract that we are developing) from our frontend. So, we didn't fix this point."

VEE-01: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	BondVsErc20Exchange.sol L234-L238, L280-L284

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true` , many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase to the Bond Token.

Client Comment:
"We didn’t fix this point for BondToken transfer because error message will be more confusing if use safeTransfer."

VEE-02: Lack of address verification

Type	Severity	Location
Volatile Code	Medium	BondVsErc20Exchange.sol L130-L132, L152-L153, L175-L178, L358-L361, L394-395

Description:

The linked function implementations do not properly sanitize the `address` variables they utilize.

Recommendation:

Additional sanitization should be imposed to ensure proper operation of the system i.e. inequality checks with the zero address.

Alleviation:

Issue Resolved

VEE-03: Lack of address verification for passed `bondPricerAddress`.

Type	Severity	Location
Volatile Code	Major	BondVsErc20Exchange.sol L132, L153

Description:

There are no direct checks that the underlying code of the Bond Pricer contract is valid for a particular input Bond Pricer address.

Recommendation:

We recommend that the Bond Pricers that should be acceptable by the Exchange are generated via a factory that Lien provides, thus allowing an easy validation to occur by ensuring the contract was deployed via the factory and consequently has unaltered bytecode.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase.

Client comment:
"Users will use the pool managed by Lien team (pools created by our address or Aggregator contract that we are developing) from our frontend. So, we didn’t fix this point."

VEE-04: Lack of restrictions for `deleteVsErc20Pool`

Type	Severity	Location
Control Flow	Major	BondVsErc20Exchange.sol L164-L166

Description:

Anyone knowing a pool's ID could call `deleteVsErc20Pool` function and delete it as there is no ACL imposed.

Recommendation:

An ACL check should be imposed that verifies the caller of the deletion operation is the bond pool owner.

Alleviation:

Issue Resolved.

BVE-01: Lack of address verification

Type	Severity	Location
Volatile Code	Medium	BondVsEthExchange.sol L125-L126, L145-L146, L168-L170

Description:

The linked function implementations do not properly sanitize the `address` variables they utilize.

Recommendation:

Additional sanitization should be imposed to ensure proper operation of the system i.e. inequality checks with the zero address.

Alleviation:

Issue Resolved.

BVE-02: Lack of address verification for passed `bondPricerAddress`.

Type	Severity	Location
Volatile Code	Major	BondVsEthExchange.sol L126, L146

Description:

There are no direct checks that the underlying code of the Bond Pricer contract is valid for a particular input Bond Pricer address.

Recommendation:

We recommend that the Bond Pricers that should be acceptable by the Exchange are generated via a factory that Lien provides, thus allowing an easy validation to occur by ensuring the contract was deployed via the factory and consequently has unaltered bytecode.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase.

Client comment:
"Users will use the pool managed by Lien team (pools created by our address or Aggregator contract that we are developing) from our frontend. So, we didn't fix this point."

BVE-03: Requisite Value of ERC-20 `transferFrom()` / `transfer()` Call

Type	Severity	Location
Logical Issue	Minor	BondVsEthExchange.sol L238, L280

Description:

While the ERC-20 implementation does necessitate that the `transferFrom()` / `transfer()` function returns a `bool` variable yielding `true` , many token implementations do not return anything i.e. Tether (USDT) leading to unexpected halts in code execution.

Recommendation:

We advise that the `SafeERC20.sol` library is utilized by OpenZeppelin to ensure that the `transferFrom()` / `transfer()` function is safely invoked in all circumstances.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase to the Bond Token.

Client Comment:
"We didn’t fix this point for BondToken transfer because error message will be more confusing if use safeTransfer."

BVE-04: Lack of restrictions for `deleteVsEthPool`

Type	Severity	Location
Control Flow	Major	BondVsEthExchange.sol L157-L159

Description:

Anyone knowing a pool's ID could call `deleteVsEthPool` function and delete it as there is no ACL imposed.

Recommendation:

An ACL check should be imposed that verifies the caller of the deletion operation is the bond pool owner.

Alleviation:

Issue resolved.

DTL-01: Divide before multiply

Type	Severity	Location
Mathematical Operations	Minor	DateTimeLibrary.sol L31-L52, L71-L96

Description:

Solidity integer divisions can truncate. As a result, performing multiplication before divison might reduce precision.

Recommendation:

Consider ordering multiplication before division if the result of the multiplication permits.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase as they say it's intended functionality.

Client comment:
"We intentionally truncate number, so didn’t fix this point."

DTL-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	DateTimeLibrary.sol L1

Description:

The contract has unlocked compiler version. 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.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue Resolved

DIG-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Digits.sol L1

Description:

The contract has unlocked compiler version. 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.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue Resolved

BNI-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	BondTokenNameInterface.sol L1

Description:

The contract has unlocked compiler version. 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.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue Resolved

OIE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	OracleInterface.sol L1

Description:

The contract has unlocked compiler version. 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.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

Issue Resolved

BMR-01: Incompatible Structure

Type	Severity	Location
Logical Issue	Major	BondMaker.sol L103

Description:

`unzipLineSegment` in `DetectBondShape.sol` at line 74 returns an array of 4 `uint64` variables whereas it is used in `BondMaker.sol` at line 103 to create an element in `_registeredFnMap[fnMapID]` which should be a `LineSegment` struct.

Recommendation:

We recommend the return type of `unzipLineSegment` is changed.

Alleviation:

Issue Resolved

BMR-02: Missing Requirement

Type	Severity	Location
Logical Issue	Minor	BondMaker.sol L107

Description:

The function `registerNewBond` does not evaluate whether the rightmost x-coordinate is equal to the maturity timestamp.

Recommendation:

This check should be imposed by the codebase.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase as they say it's intended functionality.

BMR-03: Inefficient Search

Type	Severity	Location
Gas Optimization	Informational	BondMaker.sol L155

Description:

The array of `rateBreakPoints` can become a sorted array to allow faster break point existence evaluation.

Recommendation:

The search should be optimized with a sorted array.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase.

POL-01: Redundant Statements

Type	Severity	Location
Dead Code	Informational	Polyline.sol L65, L66, L108, L109

Description:

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation:

We advise that they are removed to better prepare the code for production environments.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase as they say it's intended functionality.

Client comment:

"This check is needed for bond shape detection (detection of SBT etc.), not for bond liquidation, so this part of code should be in either `registerNewbond()` or `DetectBondShape.sol`. `DetectBondShape.sol` is called for every time the bond is sold. For saving gas, we put this part in `Polyline.sol`."

POL-02: Potentially Redundant Functions

Type	Severity	Location
Gas Optimization	Medium	Polyline.sol L51, L60

Description:

Instead of saving the segments and check that the segments are connected, we can just store the breakpoints of the polyline. In this case we don't need to use the function `assertLineSegment` and `assertPolyline`.

Recommendation:

We advise that points are stored instead of line segments within the system to prevent data duplication.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase as this would require very large modification to the code and that could cause a new and fatal error.

GPG-01: Inefficient Variable Type

Type	Severity	Location
Gas Optimization	Informational	GeneralizedPricing.sol L230, L231, L232, L233

Description:

Since we require `spotPrice, strikePrice, volatilityE8, untilMaturity` to be greater than zero we can store them in a `uint256` instead of an `int256`.

Recommendation:

We recommend changing the variable type according to the issue's description.

Alleviation:

The development team has acknowledged this exhibit but did not make any changes to the codebase as they opted to unify the type of numbers to `int256`.

GPG-02: Integer Overflow / Underflow

Type	Severity	Location
Mathematical Operations	Medium	GeneralizedPricing.sol L205

Description:

The multiplication in the mathematical expression can overflow. Moreover, `lbtPrice` should be non-negative but in edge cases can be assigned negative values by the formula on the right-hand side.

Recommendation:

We recommend using `SafeMath` for these mathematical operations to ensure edge cases are accounted for.

Alleviation:

After discussion with the Lien team, we concluded that this exhibit is rendered null by checks on L274-L289 on GeneralizedPricing.sol.

Client comment:
"Change visibility from `public` to `internal` and add comment. (The possibility of overflow is already checked before this function is called)."

GPG-03: Incorrect Comment for `require`

Type	Severity	Location
Logical Issue	Informational	GeneralizedPricing.sol L119

Description:

In the function `_calcSbtShapePrice` three points are needed but the comment denotes only two.

Recommendation:

The comment should be changed accordingly.

Alleviation:

Issue Resolved.

DBS-01: Missing Requirement

Type	Severity	Location
Logical Issue	Medium	DetectBondShape.sol L94, L119, L179

Description:

In the functions `_isLBTShape(), _isTriangle(), _isSBT()` the input points are not verified whether they indeed form a polyline. These functions are used in `getBondType` and it is not checked there either. We already have a function for it called `assertPolyline()`.

Recommendation:

The arguments should be validated via the `assertPolyline()` function to prevent unexpected behaviour.

Alleviation:

Issue Resolved.

Client comment:
"We produced two functions. One gets fnMap from bondMaker (hence no need to pass `assertPolyline()`), the other one gets fnMap as an argument and revert transaction if fnMap doesn't pass `assertPolyline()`. `BondExchange` uses the former function."

Appendix

Finding Categories

Gas Optimization

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

Mathematical Operations

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

Logical Issue

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

Control Flow

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

Volatile Code

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

Data Flow

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

Language Specific

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

Coding Style

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

Inconsistency

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

Magic Numbers

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

Compiler Error

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

Dead Code

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

Icons explanation

✓ : Issue resolved

ⓘ: Issue not resolved. The team will be fixing the issues in the own timeframe.

: Issue partially resolved. Not all instances of an issue was resolved / Issue acknoweldged won't resolve