Ocean Protocol

Security Assessment

October 8th, 2020

For :
Ocean Protocol

By :
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

Overview

Project Summary

Project Name	Ocean Protocol
Description	Ocean Protocol is a decentralized data marketplace implementation building on DeFi space principles.
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commits	1. 274e21c4c2792515bd631a673b7564ddf22abfe0 2. 17ad71aa78ad9f4bab2f4fd46fa8e3b28ce06f93

Audit Summary

Delivery Date	Oct. 8, 2020
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	Aug. 11, 2020 - Oct. 8, 2020

Vulnerability Summary

Total Issues	60
Total Critical	0
Total Major	1
Total Minor	7
Total Informational	52

Executive Summary

Our audit approached the codebase from a simultaneous security and due diligence perspective to ensure that the code output of the Ocean Protocol is of the highest quality.

While most of the issues we pointed out were optimizational, we identified 7 minor and 1 major exhibits that needed to be attended to. Of those findings, OPF-01 and BPO-14 were minor findings that relate to ERC-20 compatibility. The Ocean Protocol chose to not apply them, rendering their fee acquisition process incompatible with ERC-20 tokens that do not return a `bool` variable, such as Tether (USDT), an issue that was present in the V1 implementation of Uniswap.

Additionally, BPO-04 hints at a misunderstanding of how contract data is accessed at the EVM level rather than the Solidity level, potentially enabling attack vectors based on market knowledge that should otherwise be inaccessible on-chain.

We strongly advise that all findings that are minor severity and above are tended to in order to ensure that the Ocean Protocol v3 implementation is secure to a great extent.

DTF-01: Inefficient Greater-Than Comparisons w/ Zero

Type	Severity	Location
Optimization	Informational	DTFactory.sol L78

Description:

The linked code statements conduct a greater-than comparison between zero and an unsigned integer.

Recommendation:

As unsigned integers are restricted to the non-negative range, it is possible to instead conduct an inequality comparison with zero optimizing the gas cost necessary.

Alleviation:

The team opted to consider our references and changed to inequality comparison.

DTF-02: Redundant Event Variable

Type	Severity	Location
Optimization	Informational	DTFactory.sol L36

Description:

The linked `TokenRegistered` event includes the `block.number` within the variables it emits.

Recommendation:

As this variable is included indirectly along with the event's emittence, it is instead possible to derive it from metadata and thus can be considered redundant. As such, we advise its removal.

Alleviation:

The team opted to consider our references and removed the redundant variable `registeredAt` from the event.

DTF-03: Necessity of Function Visibility

Type	Severity	Location
Optimization	Informational	DTFactory.sol L74

Description:

The `createToken` function is declared as `public` yet remains unused by any of the contracts within the project.

Recommendation:

If it is envisioned to only be externally called, it is advisable to instead set its visibility to `external` and convert its `string` type variables from `memory` to `calldata` thus optimizing the overall gas cost of the function.

Alleviation:

The case was a situational and no alleviations were applied.

DTF-04: User Defined Getter

Type	Severity	Location
Optimization	Informational	DTFactory.sol L22, L113-L119, L20, L121-L127

Description:

In general, it is a good practise to avoid user-defined getters and instead rely on compiler-generated ones via the `public` keyword as they are far more maintainable and aid in the legibility of the codebase.

Recommendation:

As the `getCurrentTokenIndex` and `getTokenTemplate` getters conduct no special operations on the variables they are meant to return, we advise that the variables themselves are declared as `public` and the getters are omitted.

Alleviation:

The case was a situational and no alleviations were applied.

DTF-05: Invalid Documentation

Type	Severity	Location
Coding Style	Informational	DTFactory.sol L113-L116

Description:

The `getCurrentTokenIndex` function, as its comments dictate, should return the index of the current token. However, the variable returned is actually initialized at `1` and is named `currentTokenCount`.

Recommendation:

As this appears to not be a conventional zero-based index, we advise that the naming convention as well as comments utilized are rephrased to accurately represent what the function does.

Alleviation:

The case was a situational and no alleviations were applied.

DTF-06: Unlocked Compiler Version

Type	Severity	Location
Optimization	Informational	DTFactory.sol L1

Description:

The compiler version utilized in this file uses the "^" prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts.

It is a general practise to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily.

Recommendation:

We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team opted to consider our references and locked the compiler version to `0.5.7`.

BCO: General Comment

Type	Severity	Location
Coding Style & Optimization	Informational	BConst.sol

Description:

No findings were identified at this stage of the audit.

Recommendation:

However, we advise that comments are introduced at each declaration to properly represent what they are meant to depict. Additionally, while the shorthand `uint` is acceptable by the compiler we advise that its full format `uint256` is used instead to aid in the legibility of the codebase.

Alleviation:

The case was a situational and no alleviations were applied.

BFA-01: Inconsistent Error Handling

Type	Severity	Location
Coding Style	Informational	BFactory.sol L40, L52

Description:

Throughout the contracts of the project proper sanitization of input variables during construction is conducted whereby the addresses of contracts are ensured to be different than zero. However, the error message provided is of a completely different convention than `DTFactory.sol` for instance.

Recommendation:

We advise that error handling is streamlined across the codebase.

Alleviation:

The team opted to consider our references and changed the error messages in `DTFactory.sol` to match the codebase's pattern.

BFA-02: User Defined Getters

Type	Severity	Location
Optimization	Informational	BFactory.sol L22, L69-L77

Description:

In general, it is a good practise to avoid user-defined getters and instead rely on compiler-generated ones via the `public` keyword as they are far more maintainable and aid in the legibility of the codebase.

Recommendation:

As the `getBPool` getter conduct no special operations on the variables it is meant to return, we advise that the variable is declared as `public` and the getter is omitted.

Alleviation:

The team opted to consider our references and changed visibility specifier of the variable `bpoolTemplate` and removed the custom getter function `getBPool`.

BFA-03: Redundant Event Variable

Type	Severity	Location
Optimization	Informational	BFactory.sol L32

Description:

The linked `BPoolRegistered` event includes the `block.number` within the variables it emits `indexed` as well.

Recommendation:

As this variable is included indirectly along with the event's emittence, it is instead possible to derive it from metadata and thus can be considered redundant. As such, we advise its removal.

Alleviation:

The team opted to consider our references and removed the redundant variable `registeredAt` from the event.

BFA-04: Unlocked Compiler Version

Type	Severity	Location
Optimization	Informational	BFactory.sol L1

Description:

The compiler version utilized in this file uses the "^" prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts.

It is a general practise to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily.

Recommendation:

We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team opted to consider our references and locked the compiler version to `0.5.7`.

BMA-01: Explicit `return` of Named Return Variable

Type	Severity	Location
Optimization	Informational	BMath.sol L42, L73, L104, L143, L181, L229, L279

Description:

The `spotPrice` is a named return variable, meaning that whatever value it holds will be automatically returned at the end of the function's execution. On the last statement of `calcSpotPrice`, the final value the function is meant to return is assigned to the return variable as well as returned explicitly.

Recommendation:

We advise that either the explicit `return` statement is removed or that the assignment as well as the explicitly named return variable are omitted, the latter of which we advise.

Alleviation:

The case was a situational and no alleviations were applied.

BMA-02: Inconsistent Specification

Type	Severity	Location
Contract Design	Minor	BMath.sol L180

Description:

The specification of the `calcSingleInGivenPoolOut` function states that the denominator of the division is equal to 1, denoted by the constant `BONE`, subtracted by the normalized weight (weight in divided by total weight) and finally multiplied by the swap fee.

This is not aligned with the actual calculations carried within the function as a final additional subtraction occurs between the final result of the above equation and once again the value of 1 denoted by `BONE`, the value being subtracted from `BONE`.

Recommendation:

We advise that either the code statements or the documentation are updated to reflect this.

Alleviation:

No alleviations were applied despite the severity of the exhibit.

BMA-03: Variable Naming Convention

Type	Severity	Location
Coding Style	Informational	BMath.sol

Description:

While the documentation of the functions was extremely helpful in properly evaluating the intended purpose of the functions, the variable names utilized within (foo, bar, zaz etc.) are generally utilized as placeholders and are ill-advised to exist in production-ready code.

Recommendation:

We advise that potentially meaningful names are utilized for these variables as they do not affect the generated bytecode.

Alleviation:

The case was a situational and no alleviations were applied.

BMA-04: Calculation Optimizations

Type	Severity	Location
Optimization	Informational	BMath.sol

Description:

The whole contract could be further optimized.

Recommendation:

Firstly, in multiple sections of the functions in-memory variables can be re-assigned and re-used to avoid declaring a new in-memory variable and thus optimizing the final gas cost of the functions.

Lastly, it is possible to create `internal` functions for certain common formulas between the granded mathematical equations, such as the calculation of `1 - ((1 - (tO / tW)) * sF`, w/ tO /tI being an input variable of the `internal` function. This would optimize the final bytecode size of the contract.

Alleviation:

The case was a situational and no alleviations were applied.

BNU-01: Documentation of Functionality

Type	Severity	Location
Coding Style	Informational	BNum.sol L69-L71, L82-L84

Description:

The functions `bmul` and `bdiv` conducts a ceiling operation on the result which is undocumented.

Recommendation:

We advise that comments preceding the function are added that detail this side-effect of the function's multiplication in contrast to Solidity's traditional flooring operation. Overall, the function appears to be a fork of wad and ray based math based on the DSMath library albeit with a different unitary representation in `BONE`.

Alleviation:

The case was a situational and no alleviations were applied.

BNU-02: Incorrect Comment

Type	Severity	Location
Coding Style	Informational	BNum.sol L142

Description:

The linked comment contains a discrepancy with the actual implementation that accompanies it. The comment implementation contains a subtraction with 1 whereby the Solidity implementation contains an addition.

Recommendation:

We advise that the linked comment is changed to the following:

```
//         = (product(a - i + 1, i=1-->k) * x^k) / (k!)
```

Alleviation:

The case was a situational and no alleviations were applied.

BPO-01: Struct Optimization

Type	Severity	Location
Optimization	Informational	BPool.sol L27-L28

Description:

The `index` variable is declared as a `uint` which is a shorthand of `uint256` whilst it is meant to represent values between `0` and `8`, according to the imposed limit of `BConst.sol` (`MAX_BOUND_TOKENS` being equal to `8`).

Since this number will always realistically fit even within a `uint8` representation never exceeding the value of `255` it is possible to adjust its data type to tightly pack it with the preceding `bool` variable `bound` reducing the total storage cost of the struct from 4 slots to 3, significantly optimizing gas cost.

Recommendation:

As a result, we advise that the data type of `index` is set to the remainder of subtracting the size of a `bool` from a full slot, meaning that it should be set to a `uint248`.

Alleviation:

The case was a situational and no alleviations were applied.

 BPO-02: Misleading Comment

Type	Severity	Location
Coding Style	Informational	BPool.sol L28

Description:

The comment denotes that the `index` member of the struct is `private`, a concept that does not exist within Solidity in the same terms it exists in other languages. Any contract can arbitrarily read the storage of another contract regardless of what access control restrictions are imposed, meaning that this comment can be misleading to an unaware reader.

Recommendation:

We advise that it is either removed or rephrased to better depict what it is meant to represent.

Alleviation:

The case was a situational and no alleviations were applied.

 BPO-03: Unusual Naming Convention

Type	Severity	Location
Coding Style	Informational	BPool.sol L59-L77

Description:

The naming convention utilized for the linked `modifiers` does not conform to the Solidity style guide as it includes the underscore (`_`) character on both ends of the name declaration.

Recommendation:

We advise that the naming convention is refactored to align with that of the Solidity style guide. For this purpose, a strict linter may prove helpful.

Alleviation:

The case was a situational and no alleviations were applied.

 BPO-04: Invalid Assumption

Type	Severity	Location
Volatile Code	Major	BPool.sol L74-L77, L231, L239, L248, L258, L266, L277, L287, L295, L458, L476

Description:

The `_viewlock_` modifier and the way it is utilized infer that the functions it guards will be uninvokable when the reentrancy mutex has been placed on the contract. While this is true, they will not be "unviewable" as it would still be possible for an attacker to gain access to the underlying data the function's protect via low level assembly calls.

As such, its usefulness can be disputed as it can ultimately be bypassed.

Recommendation:

We advise that its intended purpose is evaluated and that the concerns that led to its implementation are shared with us so we can provide better insight as to how this can be tackled.

Alleviation:

No alleviations were applied despite the severity of the exhibit.

 BPO-05: Variable Ordering

Type	Severity	Location
Optimization	Informational	BPool.sol L79-L96

Description:

Within Solidity, the order of variable declaration is important within a contract as it dictates how the compiler will tightly-pack the variables it is provided with. While the layout of the contract does not appear to follow a particular convention, it has tightly packed most slots possible except one.

Recommendation:

We advise that the L88 declaration of `_finalized` is relocated past the L96 declaration of `initialized` to have those two `bool` data types tight packed into the same slot.

Alleviation:

The case was a situational and no alleviations were applied.

BPO-06: Redundant Value Assignment

Type	Severity	Location
Optimization	Informational	BPool.sol L79-L96

Description:

The linked `initialized` variable is a `bool` data type. In Solidity, all data types are automatically assigned their default value, in the case of a `bool` being `false`, meaning that the explicit assignment of `false` to the variable is redundant.

Recommendation:

We advise that the variable assignment is removed and only the variable declaration remains in place.

Alleviation:

The case was a situational and no alleviations were applied.

BPO-07: Variable Usefulness

Type	Severity	Location
Optimization	Informational	BPool.sol L119-L120

Description:

Throughout the codebase, these two input variables appear to be hard-coded as `false` when passed to the function call.

Recommendation:

We advise that their purpose is evaluated and if deemed unnecessary proper adjustments are made to the initializer to avoid gas-costly unnecessary assignments.

Alleviation:

The case was a situational and no alleviations were applied.

BPO-08: User Defined Getters

Type	Severity	Location
Optimization	Informational	BPool.sol L202-L214, L256-L262, L285-L299,

Description:

In general, it is a good practise to avoid user-defined getters and instead rely on compiler-generated ones via the `public` keyword as they are far more maintainable and aid in the legibility of the codebase.

Recommendation:

As the linked getters conduct no special operations on the variables they are meant to return, we advise that the variables themselves are declared as `public` and the getters are omitted.

Alleviation:

The case was a situational and no alleviations were applied.

BPO-09: Functionally Equivalent Getters

Type	Severity	Location
Optimization	Informational	BPool.sol L230-L244

Description:

The two linked getters return the exact same variables albeit the latter of the two imposes a `require` check prior to fulfilling the getter request.

Recommendation:

As these getters are identical, they redundantly increase the bytecode of the contract and thus one of the two can be omitted as the latter can be replaced by a combination of `isFinalized` and `getCurrentTokens`.

Alleviation:

The case was a situational and no alleviations were applied.

BPO-10: `require` to `modifier`

Type	Severity	Location
Optimization	Informational	BPool.sol L242, L252, L306, L307

Description:

The linked `require` statements are replicated multiple times across the contact and can instead be set as re-usable `modifiers`.

Recommendation:

We advise that they are indeed done so to increase the legibility and maintainability of the codebase.

Alleviation:

The case was a situational and no alleviations were applied.

BPO-11: Incorrect Variable Labels

Type	Severity	Location
Coding Style	Informational	BPool.sol L313-L324

Description:

The function `setController` is meant to replace the existing controller address with a new one, however the new one is labelled as `manager` and its accompanying error message also states `ERR_INVALID_MANAGER_ADDRESS` which is not the case as we are handling a controller at this point.

Recommendation:

We advise that a proper naming convention is utilized for this variable and error.

Alleviation:

The case was a situational and no alleviations were applied.

BPO-12: Pull-over-Push Pattern

Type	Severity	Location
Volatile Code	Minor	BPool.sol L313-L324

Description:

Directly replacing the managerial address of a contract is dangerous as a single misinputed character can freeze the administrative functions of the contract indefinitely.

Recommendation:

We advise that the pull-over-push pattern is applied here whereby a new controller would be proposed that would subsequently need to accept the proposal, signifying that access to the private key of the address does indeed exist.

Alleviation:

No alleviations were applied despite the severity of the exhibit.

 BPO-13: `delete` Instead of Empty Assignment

Type	Severity	Location
Optimization	Informational	BPool.sol L435-L440

Description:

The linked statement assigns a zeroed out struct to the mapping location of the unbound token.

Recommendation:

A `delete` operation would be equivalent and would properly refund the gas occupied by the storage block instead of an instantiation and assignment which is more gas costly.

Alleviation:

The case was a situational and no alleviations were applied.

 BPO-14: ERC-20 Compatibility Notice

Type	Severity	Location
Volatile Code	Minor	BPool.sol L879-L891

Description:

The linked function implementations are correct in evaluating whether the transfers were successful or not, however certain ERC20 tokens incorrectly implement the specification by ultimately not returning any variable.

Recommendation:

For maximum token support, we advise a SafeERC20 wrapper from OpenZeppelin is utilized instead.

Alleviation:

No alleviations were applied despite the severity of the exhibit.

BPO-15: Redundant Type Casting

Type	Severity	Location
Optimization	Informational	BPool.sol L565-L566, L639-L640

Description:

The variables utilized are already of type `address` and are redundantly casted to that data type.

Recommendation:

We advise these unnecessary casts are omitted.

Alleviation:

The case was a situational and no alleviations were applied.

BTO-01: Redundant `require` Checks

Type	Severity	Location
Optimization	Informational	BToken.sol L52-L55, L62, L144

Description:

The linked `require` checks are redundant as they are checked by the underlying implementation of `bsub`.

Recommendation:

We advise to remove the unnecessary checks. Also, if a custom error message is instead desired, the `BNum.sol` implementation should be expanded to support those.

Alleviation:

The case was a situational and no alleviations were applied.

BTO-02: Variable Mutability Specifiers

Type	Severity	Location
Optimization	Informational	BToken.sol L79-L81

Description:

The linked variables are only assigned to once in their actual declaration.

Recommendation:

We advise to set them to `constant` to significantly reduce the gas cost involved in utilizing them.

Alleviation:

The case was a situational and no alleviations were applied.

BTO-03: User Defined Getters

Type	Severity	Location
Optimization	Informational	BToken.sol L40, L103-L105

Description:

In general, it is a good practise to avoid user-defined getters and instead rely on compiler-generated ones via the `public` keyword as they are far more maintainable and aid in the legibility of the codebase.

Recommendation:

As the linked getters conduct no special operations on the variables they are meant to return, we advise that the variables themselves are declared as `public` and the getters are omitted.

Alleviation:

The case was a situational and no alleviations were applied.

OPF-01: ERC-20 Compatibility Notice

Type	Severity	Location
Volatile Code	Minor	OPFCommunityFeeCollector.sol L73-L79

Description:

The linked function implementations are correct in evaluating whether the transfers were successful or not, however certain ERC20 tokens incorrectly implement the specification by ultimately not returning any variable.

Recommendation:

For maximum token support, we advise a SafeERC20 wrapper from OpenZeppelin is utilized instead.

Alleviation:

No alleviations were applied despite the severity of the exhibit.

FRE-01: Struct Optimization

Type	Severity	Location
Optimization	Informational	FixedRateExchange.sol L19-L25

Description:

The `Exchange` struct can be further optimized.

Recommendation:

We advise to reorder the variable declaration of `bool active` before the `uint256 fixedRate`, thus tight packing the `bool` variable with one of the `address` variables reducing the number of slots necessary by the struct from 5 to 4.

Alleviation:

The team opted to consider our references and changed the order members of the `Exchange` struct are packed.

FRE-02: Visibility Specifiers Missing

Type	Severity	Location
Coding Style	Informational	FixedRateExchange.sol L28-L29

Description:

The visibility specifiers for the linked variables should be explicitly set as it is standard security practise and aids in the readability of the codebase.

Recommendation:

We advise to set the visibility specifiers for the linked variables.

Alleviation:

The team opted to consider our references and declared both of the linked variables as `private`.

FRE-03: Empty Constructor

Type	Severity	Location
Optimization	Informational	FixedRateExchange.sol L88

Description:

Empty constructors are redundant as they are directly implied even if not provided.

Recommendation:

We advise that the linked constructor is removed.

Alleviation:

The team opted to consider our references and removed the empty constructor.

FRE-04: Inefficient Greater-Than Comparisons w/ Zero

Type	Severity	Location
Optimization	Informational	FixedRateExchange.sol L118, L122-L123, L221, L277

Description:

The linked code statements conduct a greater-than comparison between zero and an unsigned integer.

Recommendation:

As unsigned integers are restricted to the non-negative range, it is possible to instead conduct an inequality comparison with zero optimizing the gas cost necessary.

Alleviation:

The team opted to consider our references and changed to inequality comparisons for the linked statements.

FRE-05: Misleading Event Variable Names

Type	Severity	Location
Coding Style	Informational	FixedRateExchange.sol L71, L155, L310, L77, L335

Description:

The event declarations state that a `timestamp` type variable is emitted along the event whereas a `block.number` is provided instead of a `block.timestamp` wherever the events are emitted.

Recommendation:

We advise that either the variable is completely omitted from the event if representing a `block.number`, or that a timestamp is properly emitted by the event.

Alleviation:

The team opted to consider our references and removed the redundant variables from the events.

FRE-06: Redundant Utilization of `abi.encodePacked`

Type	Severity	Location
Optimization	Informational	FixedRateExchange.sol L176-L180

Description:

The `abi.encodePacked` function solely makes sense when the variables it operates on can be tightly packed.

Recommendation:

As the current input variables are of type `address` which is equivalent to 160-bits, none of the input variables can be tightly packed and thus the usage of `abi.encode` is more optimal. Additionally, when generating identifiers it is ill-advised to use packing mechanisms as they can lead to identifier collissions.

Alleviation:

The team opted to consider our references and used the `abi.encode` function.

FRE-07: Activate / Deactivate to Toggle Function

Type	Severity	Location
Optimization	Informational	FixedRateExchange.sol L289-L337

Description:

The linked functions contain identical statements with the core differentiator being the boolean literal they utilize.

Recommendation:

We advise that they are merged into a single function that toggles the variable to reduce the total bytecode of the contract.

Alleviation:

The team opted to consider our references and implemented a single `toggleExchangeState` function.

FRE-08: Duplicate Mapping Lookups

Type	Severity	Location
Optimization	Informational	FixedRateExchange.sol L404-L408

Description:

All members of the `Exchange` struct are accessed and each mapping lookup operation as well as data retrieval costs significant gas.

Recommendation:

As all members of the struct are accessed, it is possible to instead assign the result of the lookup operation to a `memory` declaration of the `Exchange` struct that is subsequently accessed for the return variables.

Alleviation:

The team opted to consider our references and stored the instance of the `Exchange` struct to `memory` before accessing the struct members.

FRE-09: Unlocked Compiler Version

Type	Severity	Location
Optimization	Informational	FixedRateExchange.sol L1

Description:

The compiler version utilized in this file uses the "^" prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts.

It is a general practise to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily.

Recommendation:

We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team opted to consider our references and locked the compiler version to `0.5.7`.

DDO-01: Empty Constructor

Type	Severity	Location
Optimization	Informational	DDO.sol L45

Description:

Empty constructors are redundant as they are directly implied even if not provided.

Recommendation:

We advise that the linked constructor is removed.

Alleviation:

The team completely removed the `DDO.sol` file.

DDO-02: Unlocked Compiler Version

Type	Severity	Location
Optimization	Informational	DDO.sol L1

Description:

The compiler version utilized in this file uses the "^" prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts.

It is a general practise to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily.

Recommendation:

We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team completely removed the `DDO.sol` file.

DTT-01: Variable Ordering

Type	Severity	Location
Optimization	Informational	DataTokenTemplate.sol L19

Description:

Within Solidity, the order of variable declaration is important within a contract as it dictates how the compiler will tightly-pack the variables it is provided with. While the layout of the contract does not appear to follow a particular convention, it has tightly packed most slots possible except one.

Recommendation:

We advise that the L19 declaration of `initialized` is relocated past the L25 declaration of `_minter` to have the `bool` and `address` data types tight packed into the same slot.

Alleviation:

The team opted to consider our references and relocated the linked variable to the correct position.

DTT-02: Inexistent Indexing

Type	Severity	Location
Optimization	Informational	DataTokenTemplate.sol L30-L47

Description:

The linked `event` declarations contain no `indexed` variables.

Recommendation:

We advise that `indexed` variables are introduced as they greatly optimize the speed of filtering a specific subset of events from the blockchain.

Alleviation:

The team opted to consider our references and introduced `indexed` variables to the linked events.

DTT-03: Inefficient Greater-Than Comparisons w/ Zero

Type	Severity	Location
Optimization	Informational	DataTokenTemplate.sol L165, L278

Description:

The linked code statements conduct a greater-than comparison between zero and an unsigned integer.

Recommendation:

As unsigned integers are restricted to the non-negative range, it is possible to instead conduct an inequality comparison with zero optimizing the gas cost necessary.

Alleviation:

The team opted to consider our references and changed to inequality comparison.

DTT-04: ERC20 Specification Incompatibility

Type	Severity	Location
Volatile Code	Minor	DataTokenTemplate.sol L24, L352-L360

Description:

The ERC20 specification denotes that the decimals of a token should be of type `uint8` whereas here both the getter as well as the storage variable are declared as `uint256`. This will cause complete incompatibility with the ERC20 specification as external contracts will not be able to properly retrieve the decimals of the contract.

Recommendation:

We advise that the variable types are properly adjusted to conform to the ERC20 specification.

Alleviation:

The team opted to consider our references and changed to data type of the `_decimals` variable to that of `uint8`.

DTT-05: Literal Assignment

Type	Severity	Location
Optimization	Informational	DataTokenTemplate.sol L169

Description:

The `_decimals` variable is being assigned to the literal `18`.

Recommendation:

The variable itself could be converted to a `constant` greatly optimizing the gas cost involved in utilizing it.

Alleviation:

The team opted to consider our references and added the `constant` mutability specifier for the `_decimals` variable.

DTT-06: Pause / Unpause to Toggle Function

Type	Severity	Location
Optimization	Informational	DataTokenTemplate.sol L294-L312

Description:

The linked functions contain identical statements with the core differentiator being the boolean literal they utilize.

Recommendation:

We advise that they are merged into a single function that toggles the variable to reduce the total bytecode of the contract.

Alleviation:

The team completely removed the pausing mechanism.

DTT-07: Pull-over-Push Pattern

Type	Severity	Location
Volatile Code	Minor	DataTokenTemplate.sol L314-L323

Description:

Directly replacing the minter address of a contract is dangerous as a single misinputed character can freeze the minting functions of the contract indefinitely.

Recommendation:

We advise that the pull-over-push pattern is applied here whereby a new minter would be proposed that would subsequently need to accept the proposal, signifying that access to the private key of the address does indeed exist.

Alleviation:

The team opted to consider our references and implemented the `proposeMinter` and `approveMinter` function, strictly following the pull-over-push pattern.

DTT-08: Unsanitized Variables

Type	Severity	Location
Volatile Code	Informational	DataTokenTemplate.sol L202-L256

Description:

The input variables of the `startOrder` function are unsanitized, meaning that anyone is capable of setting a zero `feePercentage` as well as a custom `feeCollector` address.

Recommendation:

We advise that proper sanitization of these variables is set here.

Alleviation:

The team opted to consider our references and removed the parameters `feePercentage` and `feeCollector` from the `startOrder` function while also introducing the `BASE_MARKET_FEE_PERCENTAGE` constant variable as a replacement the former one.

DTT-09: Value Based Refund

Type	Severity	Location
Volatile Code	Minor	DataTokenTemplate.sol L278-L282

Description:

The current implementation utilizes a value-based refund instead of a proportional refund.

Recommendation:

We advise that orders are somehow stored and are subsequently accessed here to enable proportional refunds instead of value based refunds, as a refund could possibly exceed the original order's amount due to no sanitization being in place.

Alleviation:

No alleviations were applied despite the severity of the exhibit.

DTT-10: Unlocked Compiler Version

Type	Severity	Location
Optimization	Informational	DataTokenTemplate.sol L1

Description:

TThe compiler version utilized in this file uses the "^" prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts.

It is a general practise to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily.

Recommendation:

We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team opted to consider our references and locked the compiler version to `0.5.7`.

ERP-01: Unlocked Compiler Version

Type	Severity	Location
Optimization	Informational	ERC20Pausable.sol L1

Description:

The compiler version utilized in this file uses the "^" prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts.

It is a general practise to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily.

Recommendation:

We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team completely removed the `ERC20Pausable.sol` file.

DPL-01: Assembly Block Documentation

Type	Severity	Location
Optimization	Informational	Deployer.sol L35-L41

Description:

The purpose of the literals within the linked assembly block should be properly documented before proper evaluation of its functionality is conducted.

Recommendation:

We advise that proper documentation is added regarding the linked assembly code block.

Alleviation:

The team opted to consider our references and added proper documentation for the linked assebly code block.

DPL-02: Unlocked Compiler Version

Type	Severity	Location
Optimization	Informational	Deployer.sol L1

Description:

The compiler version utilized in this file uses the "^" prefix specifier, denoting that a compiler at or above the version included after the specifier should be used to compile the contracts.

It is a general practise to instead lock the compiler at a specific version rather than allow a range of compiler versions to be utilized to avoid compiler-specific bugs and be able to identify ones more easily.

Recommendation:

We recommend locking the compiler at the lowest possible version that supports all the capabilities wished by the codebase. This will ensure that the project utilizes a compiler version that has been in use for the longest time and as such is less likely to contain yet-undiscovered bugs.

Alleviation:

The team opted to consider our references and locked the compiler version to `0.5.7`.