CERTIK

Ondo

Ondo Protocol

Security Assessment

April 19th, 2021

Audited By:
Sheraz Arshad @ CertiK
sheraz.arshad@certik.org
Reviewed By:
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 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	Ondo - Ondo Protocol
Description	The Ondo Protocol implements LP tokens staking and rewarding mechanism based on tranches of underlying tokens of the liquidity pools. The underlying pool tokens are divided into senior and junior tranches with senior tranche having preference in the receiving of the LP rewards. The strategy for Uniiswap is implemented to deal with adding and removing of LP tokens from the pool contracts.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. be5f650a4984d8b8f1dca8f7a1c3827981cbac4e 2. aebd7c31d445cdb94b9edd5cc64a1ef743430d8b

Audit Summary

Delivery Date	April 19th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	April 12th, 2021 - April 19th, 2021

Vulnerability Summary

Total Issues	27
● **Total Critical**	0
● **Total Major**	0
● **Total Medium**	4
● **Total Minor**	3
● **Total Informational**	20

Executive Summary

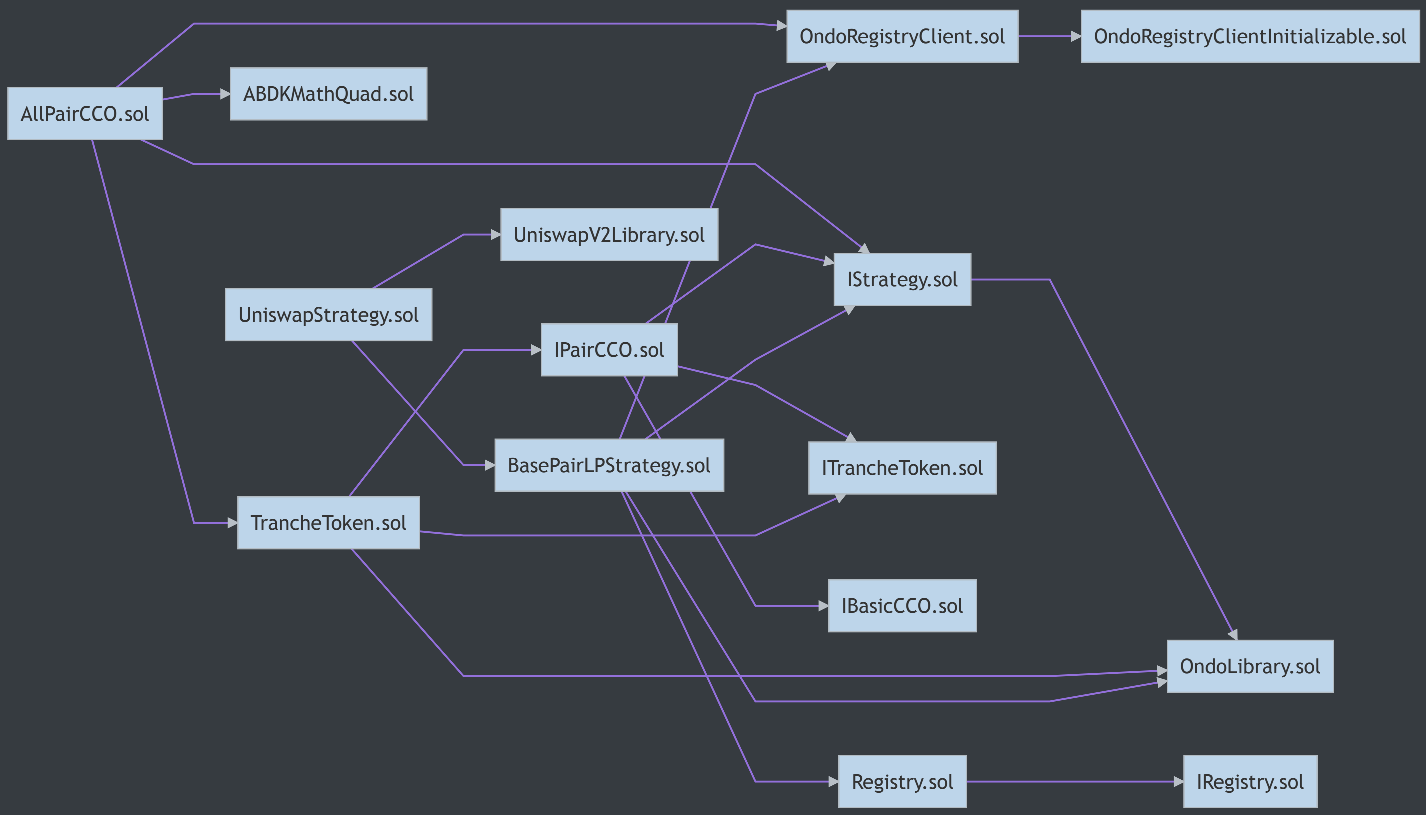
This report represents the results of CertiK's engagement with Ondo on their implementation of the Ondo Protocol smart contracts.

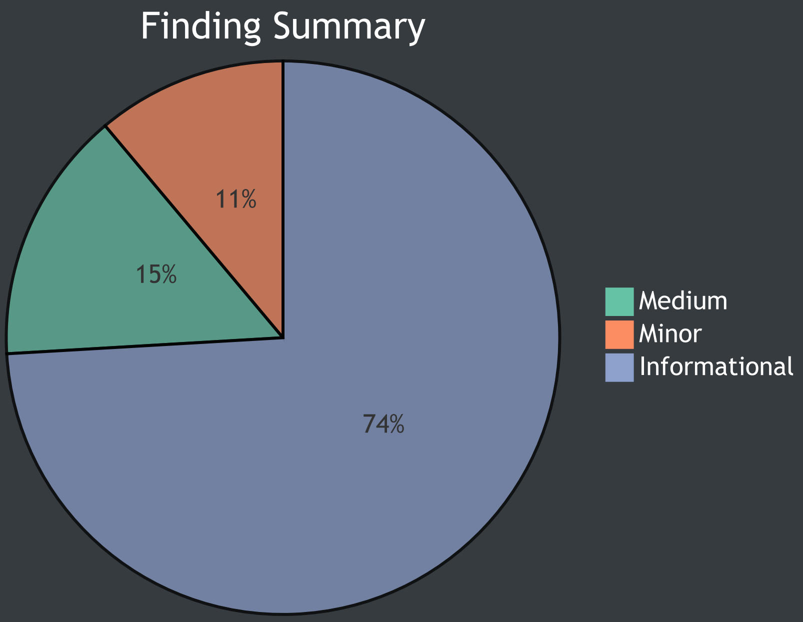
The manual and static analysis were performed in the audit. Our findings mainly refer to optimizations issues, a few minor issues and major issues. The medium issues comprise the non-checking of addresses that initialize the contracts states against zero address value and exposure of Uniswap interactions to small a probability of sandwich attacks. The minor issues comprise missing of `noPanic` check on the inherited implementations of `increaseApproval` and `decreaseApproval` functions in `TrancheToken` contract, and incorrect reflection of excess tokens in the `UniswapStrategy` contract. The remediations are applied to all of the findings except `USY-02`.

Files In Scope

ID	Contract	Location
APC	AllPairCCO.sol	contracts/AllPairCCO.sol
ORC	OndoRegistryClient.sol	contracts/OndoRegistryClient.sol
ORI	OndoRegistryClientInitializable.sol	contracts/OndoRegistryClientInitializable.sol
REG	Registry.sol	contracts/Registry.sol
TTN	TrancheToken.sol	contracts/TrancheToken.sol
IBC	IBasicCCO.sol	interfaces/IBasicCCO.sol
IPC	IPairCCO.sol	interfaces/IPairCCO.sol
IRY	IRegistry.sol	interfaces/IRegistry.sol
ISY	IStrategy.sol	interfaces/IStrategy.sol
ITT	ITrancheToken.sol	interfaces/ITrancheToken.sol
OLY	OndoLibrary.sol	libraries/OndoLibrary.sol
BPL	BasePairLPStrategy.sol	contracts/strategies/BasePairLPStrategy.sol
USY	UniswapStrategy.sol	contracts/strategies/UniswapStrategy.sol

File Dependency Graph




Finding Summary
11%
15%
74%
Medium
Minor
Informational

Manual Review Findings

ID	Title	Type	Severity	Resolved
APC-01	Lack of verification for the constructor parameter	Logical Issue	● Medium	✓
APC-02	Strategy reflects incorrect amount of excess tokens	Volatile Code	● Minor	✓
APC-03	Documentation discrepancy	Inconsistency	● Minor	✓
APC-04	Unlocked Compiler Version	Language Specific	● Informational	✓
APC-05	Inexistent Error Message	Coding Style	● Informational	✓
APC-06	Explicitly returning local variable	Gas Optimization	● Informational	✓
APC-07	Explicitly returning local variable	Gas Optimization	● Informational	✓
ORC-01	Unlocked Compiler Version	Language Specific	● Informational	✓
ORI-01	Lack of verification for the function parameter	Logical Issue	● Medium	✓
ORI-02	Unlocked Compiler Version	Language Specific	● Informational	✓
REG-01	Unlocked Compiler Version	Language Specific	● Informational	✓
REG-02	Inexistent Error Message	Coding Style	● Informational	✓
TTN-01	Functions `increaseAllowance` and `decreaseAllowance` are executable when the contract is panicked	Logical Issue	● Minor	✓
TTN-02	Unlocked Compiler Version	Language Specific	● Informational	✓
TTN-03	Inexistent Error Message	Coding Style	●	✓

			Informational	
OLY-01	Unlocked Compiler Version	Language Specific	● Informational	✓
OLY-02	Unused code	Coding Style	● Informational	✓
OLY-03	Redundant Statements	Dead Code	● Informational	✓
BPL-01	Unlocked Compiler Version	Language Specific	● Informational	✓
BPL-02	Redundant declaration of \`modifier\`	Gas Optimization	● Informational	✓
USY-01	Lack of verification for the constructor parameters	Logical Issue	● Medium	✓
USY-02	Possibility of sandwich attack	Volatile Code	● Medium	◷
USY-03	Unlocked Compiler Version	Language Specific	● Informational	✓
USY-04	Redundant assignment of state variable	Coding Style	● Informational	✓
USY-05	Unneeded use of addition assignment	Coding Style	● Informational	✓
USY-06	Unneeded read of contract's storage	Gas Optimization	● Informational	✓
USY-07	Explicitly returning local variable	Gas Optimization	● Informational	✓

APC-01: Lack of verification for the constructor parameter

Type	Severity	Location
Logical Issue	● Medium	AllPairCCO.sol L171

Description:

The constructor parameter of `_trancheTokenImpl` on the aforementioned line sets a state variable of the contract and is not validated against zero address value. If a zero address value is provided for it then it will result in unwanted state of the contract and it cannot be changed.

Recommendation:

We advise to validate the constructor parameter `_trancheTokenImpl` against zero address value to guard against setting zero address value for the contract's state variable.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

APC-02: Strategy reflects incorrect amount of excess tokens

Type	Severity	Location
Volatile Code	● Minor	AllPairCCO.sol L495

Description:

The aforementioned line returns excess tokens for a given investor and this excess amount is transferred to the user within the body of the function which results in strategy contract reflecting incorrect amount of excess tokens for a given token. Although, the current implementation does not result in any concerning vulnerability but any source reading excess tokens amounts from the strategy contract gets incorrect excess token amount.

Recommendation:

We recommend to make use of `withdrawExcess` function on the strategy contract, so the strategy contract reflects correct amounts for the excess tokens.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

CERTIK

APC-03: Documentation discrepancy

Type	Severity	Location
Inconsistency	🔵 Minor	AllPairCCO.sol L181

Description:

The comment on the aforementioned line states that the parameter `_strategist` is an EOA (Externally Owned Account) yet there is no check in the code to ensure it's an EOA address.

Recommendation:

We advise to either change the comment or implement the logic code to ensure the address is not a contract and is an EOA address.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` by changing the comment to reflect the current behaviour of the code.

APC-04: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	AllPairCCO.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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

APC-05: Inexistent Error Message

Type	Severity	Location
Coding Style	● Informational	AllPairCCO.sol L419

Description:

The linked `require` check does not contain any error message specified.

Recommendation:

We advise the error message of the check to be set properly to illustrate what the conditionals within evaluate.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

APC-06: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	● Informational	AllPairCCO.sol L479, L600, L628

Description:

The functions on the aforementioned lines explicitly return local variables which increases overall cost of gas.

Recommendation:

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

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

APC-07: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	● Informational	AllPairCCO.sol L406

Description:

The functions on the aforementioned lines explicitly return local variables which increases overall cost of gas.

Recommendation:

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

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

ORC-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	OndoRegistryClient.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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

ORI-01: Lack of verification for the function parameter

Type	Severity	Location
Logical Issue	● Medium	OndoRegistryClientInitializable.sol L48

Description:

The function parameters `_registry` sets the state variable of the contract and is not validated against zero address value. If a zero address value is provided for it then it will result in unwanted state of the contract and it cannot be changed.

Recommendation:

We recommend to validate the function parameter `_registry` against zero address value to guard against setting zero address value for the contract's state variable.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

ORI-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	OndoRegistryClientInitializable.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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

REG-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	Registry.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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

REG-02: Inexistent Error Message

Type	Severity	Location
Coding Style	● Informational	Registry.sol L30

Description:

The linked `require` check does not contain any error message specified.

Recommendation:

We advise the error message of the check to be set properly to illustrate what the conditionals within evaluate.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

TTN-01: Functions `increaseAllowance` and `decreaseAllowance` are executable when the contract is panicked

Type	Severity	Location
Logical Issue	🔵 Minor	TrancheToken.sol L76

Description:

The function `approve` on the aforementioned line reverts when the contract is in the state of `panic` yet the functions `increaseAllowance` and `decreaseAllowance` can still be used to change allowance even when the contract is in the state of `panic` .

Recommendation:

We recommend to write derived implementations of the functions `increaseAllowance` and `decreaseAllowance` in the `TrancheToken` contract where the functions revert when the contract is in the state of `panic` .

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

TTN-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	TrancheToken.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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

TTN-03: Inexistent Error Message

Type	Severity	Location
Coding Style	● Informational	TrancheToken.sol L21

Description:

The linked `require` check does not contain any error message specified.

Recommendation:

We advise the error message of the check to be set properly to illustrate what the conditionals within evaluate.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

OLY-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	OndoLibrary.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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

OLY-02: Unused code

Type	Severity	Location
Coding Style	● Informational	OndoLibrary.sol L10

Description:

The statement on the aforementioned allows the functions of `OLib` library on the type `OLib.Investor` yet the codebase does not have any instance where the library functions are called on the aforementioned type.

Recommendation:

We advise to remove the redundant statement on the aforementioned line as it is not used.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

OLY-03: Redundant Statements

Type	Severity	Location
Dead Code	● Informational	OndoLibrary.sol L32

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:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

BPL-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	BasePairLPStrategy.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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

BPL-02: Redundant declaration of `modifier`

Type	Severity	Location
Gas Optimization	● Informational	BasePairLPStrategy.sol L24

Description:

The modifier `onlyStrategist` on the aforementioned line is redundant as the same modifier is already inherited from `OndoRegistryClientInitializable` contract by the name `isStrategist` .

Recommendation:

We advise to utilize the already existing modifier in the contract instead of declaring a new one to reduce the bytecode footprint of the contract.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

USY-01: Lack of verification for the constructor parameters

Type	Severity	Location
Logical Issue	🟡 Medium	UniswapStrategy.sol L28

Description:

The constructor parameters of `_router` and `_factory` are used to set the state variables of the contract and these parameters are not validated against zero address values. If these parameters are provided as zero address values then it will result in unwanted state of the contract and they cannot be changed.

Recommendation:

We recommend validate the aforementioned parameters against zero address values to guard against setting the contract's state variables with zero address values.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

USY-02: Possibility of sandwich attack

Type	Severity	Location
Volatile Code	● Medium	UniswapStrategy.sol L189, L261

Description:

The aforementioned lines perform token swap on `uniswap`. Although, the functions are only callable by the strategist, It still possesses a possibility of sandwich attack from a malicious actor who can front-run the transaction and as `amountOutMin` is specified as 0, it will result in less than expected amount received by the contract.

Recommendation:

We recommend to provide the `amountOutMin` parameters to guard against sandwich attacks.

Alleviation:

The finding is acknowledged by the Ondo team but no alleviations are applied.

USY-03: Unlocked Compiler Version

Type	Severity	Location
Language Specific	● Informational	UniswapStrategy.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 `v0.8.3` the contract should contain the following line:

```
pragma solidity 0.8.3;
```

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

USY-04: Redundant assignment of state variable

Type	Severity	Location
Coding Style	● Informational	UniswapStrategy.sol L33

Description:

The state variable `registry` is redundantly assigned on the aforementioned line as it is already on `L52` in `OndoRegistryClientInitializable` contract.

Recommendation:

We recommend to remove the redundant assignment of variable on the aforementioned line.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b` .

USY-05: Unneeded use of addition assignment

Type	Severity	Location
Coding Style	● Informational	UniswapStrategy.sol L158

Description:

The aforementioned performs unnecessary addition assignment which can be substituted with a simple assignment to increase the legibility of the codebase as `cco_.lpTokens` will always be 0 prior to the aforementioned statement.

Recommendation:

We advise to substitute the addition assignment with a simple assignment on the aforementioned line.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

USY-06: Unneeded read of contract's storage

Type	Severity	Location
Gas Optimization	● Informational	UniswapStrategy.sol L161

Description:

The aforementioned line reads `cco_.lpTokens` from contract's storage which is inefficient as the same value is available from the local variable `lpTokens`.

Recommendation:

We advise to utilize local variable on the aforementioned line as reading from local variable costs significantly less gas than reading from the contract's storage.

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

USY-07: Explicitly returning local variable

Type	Severity	Location
Gas Optimization	● Informational	UniswapStrategy.sol L218

Description:

The function on the aforementioned line explicitly return local variables which increases overall cost of gas.

Recommendation:

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

Alleviation:

Alleviations are applied as of commit hash `aebd7c31d445cdb94b9edd5cc64a1ef743430d8b`.

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.

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.

Volatile Code

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

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.

Dead Code

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