Audit Report

MetaLend

January 2022

Github https://github.com/MetaLend-DeFi/compound-protocol

Audited by © coinscope

Table of Contents

Table of Contents 1
Contract Review 3
Audit Updates 3
Compound Protocol 4
MetaLend Variation 5
Contract Analysis 6
Supplying NFTs 7
Description 7
Recommendation 7
NFT price Feed 8
Description 8
Recommendation 8
Contract Diagnostics 9
CO - Code Optimization 10
Description 10
Recommendation 11
L05 - Unused State Variable 12
Description 12
Recommendation 12
L03 - Redundant Statements 13
Description 13
Recommendation 13
L11 - Unnecessary Boolean equality 14
Description 14
Recommendation 14
L01 - Public Function could be Declared External 15

Description 15
Recommendation 15
L04 - Conformance to Solidity Naming Conventions 16
Description 16
Recommendation 16
Contract Functions 17
AppraisalOracle 17
CErc20 18
CErc721Bridged 20
Comptroller 21
LiquidityAssessor 23
Contract Flow 24
Summary 28
Disclaimer 29
About Coinscope 30

Contract Review

The contract audit will review the files that are included in the github repository

Github	https://github.com/MetaLend-DeFi/compound-protocol
Commit	e3e5dea18d8fbf9ff2924b26cb4678774618a59c

Audit Updates

Initial Audit	26th January 2022
Corrected	

Compound Protocol

The Compound Protocol implements the mechanism of supplying or borrowing assets. In the traditional implementation, the protocol is operating on ERC20-like assets. The abstract idea is that users can deposit ERC20 assets to receive the corresponding cTokens and vice versa. cTokens is the underlying “currency” of the protocol. Essentially, cToken is the representative user’s balance. Users have the ability to mint cTokens by depositing assets or redeem cTokens by withdrawing.

The compound protocol calculates the interest for the borrowers. Interest rates fill the gap between the suppliers and the borrowers. It is universal and adjusts over time as the relationship between supply and demand changes.

There are two papers that describe the theoretical and the technical aspect of the protocol. Users can read more on the Compound Whitepaper and the Compound Protocol Specification.

An elegant implementation of the compound protocol can be found on: https://github.com/compound-finance/compound-protocol

MetaLend Variation

Metalend enriches the fundamental definition of Compound Protocol with NFTs. The basic idea is that users can operate with NFTs additional to the ERC20 assets. Essentially Metaland forked the Compound Protocol repository and added all the required steps in order to support the ERC721 protocol. Respectively to the standard cToken for the ERC20 assets, MetaLend has introduced the CErc721 for the ERC721 assets. Since the core implementation is forked by the Compound protocol, we will focus mainly on the most important changes that have been introduced.

File	SHA256
AppraisalOracle.sol	fc27ed242d41d9ec46f9029d8ffd691fd26f375f93bcfe09362c4a79cfd22110
CErc20.sol	93ab36d8230982cc7146f7f4b96b9f47d48ca08aa5d89613723f6c5f468d6921
CErc721Bridged.sol	e10eae66263582c21bd0bac0cf35506edded56d8a57593425ca06517283fe774
Comptroller.sol	2e3ae49f78efdf4f27472581eec14b2f0501e2ed352a0717e24154448c8f8839
LiquidityAssessor.sol	f152accee8aab33f2c454089859bb8de6d885bde034b6a656e10e735acc1dea2

Contract Analysis

The process of adding functionality to a well defined mechanism is challenging. Mainly there are two challenging variations in the ERC721 contracts. The first variation is to determine how the users will supply NFT assets and the second is how the compound protocol will track the NFT price.

Supplying NFTs

Description

The traditional compound protocol deposits the assets to the compound contract by transferring the corresponding amount with the standard transfer function of the ERC20 protocol.
In the MetaLend implementation, the corresponding mint function accepts an array of *tokenIds*, an array of *transactionIds* and a signature that verifies the caller's authenticity. The mint function does not verify if the *tokenIds* that are issued by the caller exist in the caller's wallet. The mint logic relies on an off-chain logic that provides a valid signature to authorize the mint.

This assumption creates vulnerability issues since the compound engine trusts an external off-chain mechanism as the source of truth. If the external source is manipulated by a hacker, then the caller will be able to apply fake assets as a reward for actual assets.

Recommendation

Could explore a solution that verifies properly the tokenIds ownership. If the NFTs are coming from different sources and networks, then a solution with verified adapters or delegators could be introduced. These adapters could validate the transaction and token ids.

NFT price Feed

Description

In the traditional compound protocol, the Price Oracle is responsible for keeping the current exchange rate of each supported asset. The price is delegated to a committee which pools prices from trusted exchanges. The exchange rate is required by the compound engine to calculate the borrowing capacity and collateral requirements. The oracle's feed is updated by four steps. Firstly, the reporters sign the price data. The reporters all well-known sources like exchanges, DeFi, etc. Secondly, the posters submit the price data to the blockchain. Then, the data are received on the on-chain side. The data is validated and stored in the data contract. Finally, when the internal application requires price data, it uses heuristics algorithms to determine a single price from the multiple resources.

MetaLend introduces NFTs as a supply and borrowing asset. The interest rate should be determined according to the NFTs price. The price tracking of NFT is not a straightforward task and it is not defined by the compound protocol whitepapers. In the current implementation of MetaLand, the caller provides a structure called *appraisal* that contains all the price feed information for the NFTs. The *appraisal* is signed by the caller. This procedure has also been ported in the CErc20 implementation. Hence, every call to the MetaLend functions requires the entire appraisal structure as argument. This methodology produces two issues:

1. It is inefficient to provide the entire price feed structure on every call.
2. The NFTs price is provided from a single one source, so it is easier to be manipulated.

Recommendation

The NFT compound protocol could implement a structure that maintains a list of prices for the NFTs. The list will be updated by the reporters and posters, similar to the Price Oracle. This way the NFTs price feed will be kept on-chain. As a result, the caller will not provide the entire price feed as an argument. Additionally, the price determination for every NFT will be more transparent and secure.

Contract Diagnostics

Critical Medium Minor

Severity	Code	Description
	CO	Code Optimization
	L05	Unused State Variable
	L03	Redundant Statements
	L11	Unnecessary Boolean equality
	L01	Public Function could be Declared External
	L04	Conformance to Solidity Naming Conventions

CO - Code Optimization

Criticality	minor
Location	LiquidityAssessor.sol#L249

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The *getAppraisal()* function is trying to find the appraisal value given a token and a token Id. The algorithm creates the appraisals array and creates two nested structures in the loop.

```
Appraisal[] memory appraisals = new Appraisal[](wire.appraisalTokens.length);

uint cursor = 0;
for (uint i = 0; i < wire.appraisalTokens.length; i++) {
    address token = wire.appraisalTokens[i];
    uint256[] memory tokenIds = new uint256[](wire.appraisalLengths[i]);
    uint[] memory values = new uint[](wire.appraisalLengths[i]);

    for (uint j = 0; j < wire.appraisalLengths[i]; j++) {
        uint256 tokenId_ = wire.appraisalTokenIds[cursor];
        uint value = wire.appraisalValues[cursor];
        tokenIds[j] = tokenId_;
        values[j] = value;
        cursor++;
    }

    appraisals[i] = Appraisal({
        token: token,
        tokenIds: tokenIds,
        values: values
    });
}

for (uint i = 0; i < appraisals.length; i++) {
    Appraisal memory appraisal = appraisals[i];
    if (appraisal.token == cErc721Token) {
        uint256[] memory tokenIds = appraisal.tokenIds;
```

```
        for (uint j = 0; j < tokenIds.length; j++) {
            if (tokenIds[j] == tokenId) {
                return appraisal.values[j];
            }
        }
    }
}
```

Recommendation

Rewrite some code segments so the runtime will be more performant.
The algorithm could directly match if the current tokenId is the desired and return.

suggestion:

```
uint cursor = 0;
for (uint i = 0; i < wire.appraisalTokens.length; i++) {
    address token = wire.appraisalTokens[i];

    for (uint j = 0; j < wire.appraisalLengths[i]; j++) {
        uint256 tokenId_ = wire.appraisalTokenIds[cursor];

        if (tokenId_ == tokenId && token === cErc721Token) {
            return wire.appraisalValues[cursor];
        }

        cursor++;
    }
}
```

L05 - Unused State Variable

Criticality	minor
Location	Comptroller.sol#L55,L52

Description

There are segments that contains unused state variable.

```
closeFactorMaxMantissa
closeFactorMinMantissa
```

Recommendation

Remove unused state variables.

L03 - Redundant Statements

Criticality	minor
Location	Comptroller.sol#L367,L284,L283 and 1 more

Description

Detect the usage of redundant statements that have no effect.

```
Comptroller
...
```

Recommendation

Remove redundant statements if they congest code but offer no value.

L11 - Unnecessary Boolean equality

Criticality	minor
Location	Comptroller.sol#L479,L727,L718 and 2 more

Description

The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
comptroller.getCollateralFactorMantissa(address(cToken)) == 0 &&
comptroller.getBorrowGuardianPaused(address(cToken)) == true &&
cToken.reserveFactorMantissa() == 1e18
require(bool,string)(msg.sender == admin || state == true,only admin can
unpause)
...
```

Recommendation

Remove the equality to the boolean constant.

L01 - Public Function could be Declared External

Criticality	minor
Location	CErc20.sol#L358,L302,L767 and 17 more

Description

Public functions that are never called by the contract should be declared external to save gas.

```
liquidateBorrowAllowedErc721
liquidateBorrowAllowed
getBlockNumber
...
```

Recommendation

Use the external attribute for functions never called from the contract

L04 - Conformance to Solidity Naming Conventions

Criticality	minor
Location	AppraisalOracle.sol#L490,L485,L58 and 25 more

Description

Solidity defines a naming convention that should be followed. Rule exceptions:

- Allow constant variable name/symbol/decimals to be lowercase.
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
_setAppraisalOracle
_setComptroller
collateralFactorMaxMantissa
...
```

Recommendation

Follow the Solidity naming convention.
https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions

Contract Functions

AppraisalOracle

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AppraisalOracle	Implementation			
	<Constructor>	Public	✓	-
	_isAppraiser	Public		-
	_setAppraiser	External	✓	-
	verifySignature	Public		-
	verifyAppraisals	External		-
	getBlockNumber	Internal		
AppraisalOracleInterface	Implementation			
	_isAppraiser	Public		-
	_setAppraiser	External	✓	-
	verifySignature	Public		-
	verifyAppraisals	External		-
AppraisalStruct	Library			

CErc20

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CompLike	Interface			
	delegate	External	✓	-
CErc20	Implementation	CToken, CErc20Interface		
	initialize	Public	✓	-
	mint	External	✓	-
	redeem	Public	✓	-
	redeemUnderlying	Public	✓	-
	borrow	Public	✓	-
	repayBorrow	External	✓	-
	repayBorrowBehalf	External	✓	-
	liquidateBorrow	Public	✓	-
	liquidateBorrowErc721Bridged	Public	✓	-
	liquidateBorrowAndRedeemErc721Bridged	Public	✓	-
	sweepToken	External	✓	-
	_addReserves	External	✓	-
	getCashPrior	Internal		
	doTransferIn	Internal	✓	
	doTransferOut	Internal	✓	
	_delegateCompLikeTo	External	✓	-
CToken	Implementation	CTokenInterface, Exponential, TokenErrorReporter		

	initialize	Public	✓	-
	transferTokens	Internal	✓	
	transfer	Public	✓	nonReentrant
	transferFrom	Public	✓	nonReentrant
	approve	External	✓	-
	allowance	External		-
	balanceOf	External		-
	balanceOfUnderlying	External	✓	-
	getAccountSnapshot	External		-
	getBlockNumber	Internal		
	borrowRatePerBlock	External		-
	supplyRatePerBlock	External		-
	totalBorrowsCurrent	External	✓	nonReentrant
	borrowBalanceCurrent	External	✓	nonReentrant
	borrowBalanceStored	Public		-
	borrowBalanceStoredInternal	Internal		
	exchangeRateCurrent	Public	✓	nonReentrant
	exchangeRateStored	Public		-
	exchangeRateStoredInternal	Internal		
	getCash	External		-
	accrueInterest	Public	✓	-
	mintInternal	Internal	✓	nonReentrant
	mintFresh	Internal	✓	
	redeemInternal	Internal	✓	nonReentrant
	redeemUnderlyingInternal	Internal	✓	nonReentrant
	redeemFresh	Internal	✓	
	borrowInternal	Internal	✓	nonReentrant
	borrowFresh	Internal	✓	
	repayBorrowInternal	Internal	✓	nonReentrant
	repayBorrowBehalfInternal	Internal	✓	nonReentrant
	repayBorrowFresh	Internal	✓	
	liquidateBorrowInternal	Internal	✓	nonReentrant
	liquidateBorrowFresh	Internal	✓	
	liquidateBorrowInternalErc721	Internal	✓	nonReentrant
	liquidateBorrowShared	Internal	✓	

	seize	External	✓	nonReentrant
	seizeInternal	Internal	✓	
	_setPendingAdmin	External	✓	-
	_acceptAdmin	External	✓	-
	_setComptroller	Public	✓	-
	_setReserveFactor	External	✓	nonReentrant
	_setReserveFactorFresh	Internal	✓	
	_addReservesInternal	Internal	✓	nonReentrant
	_addReservesFresh	Internal	✓	
	_reduceReserves	External	✓	nonReentrant
	_reduceReservesFresh	Internal	✓	
	_setInterestRateModel	Public	✓	-
	_setInterestRateModelFresh	Internal	✓	
	getCashPrior	Internal		
	doTransferIn	Internal	✓	
	doTransferOut	Internal	✓	

CErc721Bridged

Contract	**Type**	**Bases**		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CErc721Bridged	Implementation	CErc721Interface, CErc721BridgedInterface, TokenErrorReporter		
	<Constructor>	Public	✓	-
	mint	External	✓	-
	redeem	External	✓	-
	removeFromAccountTokens	Internal	✓	
	seize	External	✓	-
	seizeAndRedeem	External	✓	-
	_setComptroller	Public	✓	-
	getAccountTokens	External		-

Comptroller

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Comptroller	Implementation	ComptrollerV1Storage, ComptrollerInterface, ComptrollerErrorReporter, ExponentialNoError		
	<Constructor>	Public	✓	-
	mintAllowed	External	✓	-
	mintAllowedErc721	External	✓	-
	redeemAllowed	Public	✓	-
	redeemAllowedInternal	Internal		
	redeemAllowedErc721	Public	✓	-
	redeemVerify	External	✓	-
	borrowAllowed	External	✓	-
	repayBorrowAllowed	External	✓	-
	liquidateBorrowAllowed	External	✓	-
	liquidateBorrowAllowedErc721	External	✓	-
	seizeAllowed	External	✓	-
	seizeAllowedErc721	External	✓	-
	transferAllowed	Public	✓	-
	liquidateCalculateSeizeTokens	External		-
	_setPriceOracle	Public	✓	-
	_setCloseFactor	External	✓	-
	_setCollateralFactor	External	✓	-
	_setLiquidationIncentive	External	✓	-
	_setLiquidationDiscount	External	✓	-
	_supportMarket	External	✓	-
	_addMarketInternalErc721	Internal	✓	

	_addMarketInternal	Internal	✓	
	_setMarketBorrowCaps	External	✓	-
	_setBorrowCapGuardian	External	✓	-
	_setPauseGuardian	Public	✓	-
	_setMintPaused	Public	✓	-
	_setBorrowPaused	Public	✓	-
	_setTransferPaused	Public	✓	-
	_setSeizePaused	Public	✓	-
	_become	Public	✓	-
	isMarketListed	External		-
	getCollateralFactorMantissa	External		-
	getAllMarkets	External		-
	getAllErc721Markets	External		-
	_setCompComptroller	External	✓	-
	_setLiquidityAssessor	External	✓	-
	getBlockNumber	Public		-
	getCloseFactorMantissa	External		-
	getLiquidationDiscountMantissa	External		-
	getLiquidationIncentiveMantissa	External		-
	getBorrowGuardianPaused	External		-
	getOracle	External		-
UnitrollerAdminStorage	Implementation			
ComptrollerV1Storage	Implementation	UnitrollerAdminStorage		
Unitroller	Implementation	UnitrollerAdminStorage, ComptrollerErrorReporter		
	<Constructor>	Public	✓	-
	_setPendingImplementation	Public	✓	-
	_acceptImplementation	Public	✓	-
	_setPendingAdmin	Public	✓	-

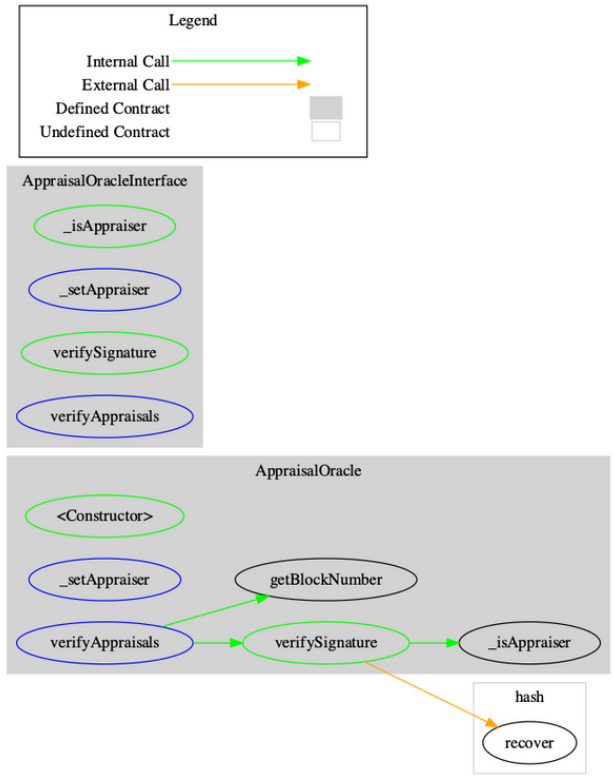
	_acceptAdmin	Public	✓	-
	<Fallback>	External	Payable	-

LiquidityAssessor

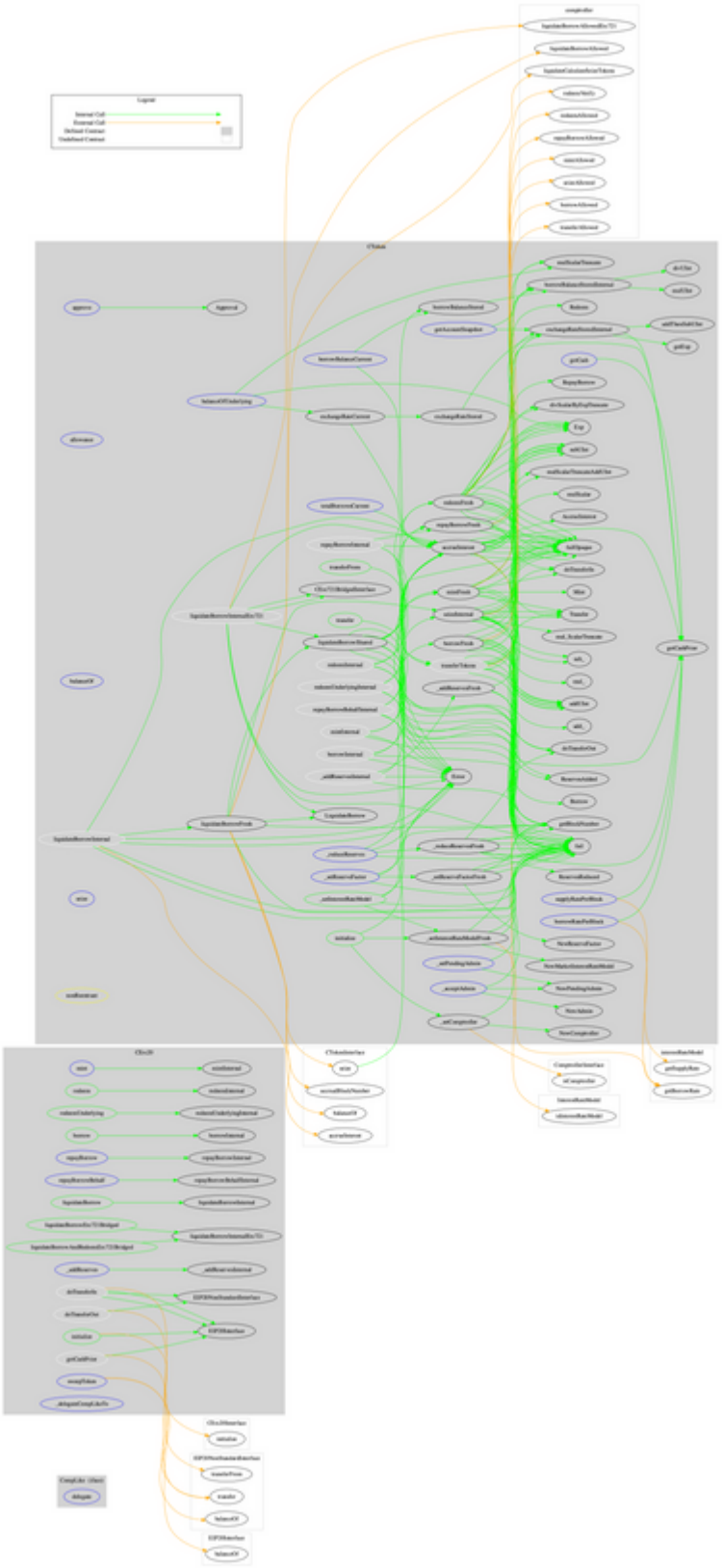
Contract	**Type**	**Bases**		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
LiquidityAssessor	Implementation	ComptrollerErrorReporter, ExponentialNoError		
	<Constructor>	Public	✓	-
	getAccountLiquidity	Public		-
	getTotalBorrows	External		-
	getHypotheticalAccountLiquidity	Public		-
	getAppraisal	Public		-
	liquidateBorrowAllowed	Public		-
	liquidateBorrowAllowedErc721	Public		-
	liquidateCalculateSeizeTokens	External		-
	isDeprecated	Public		-
	_setComptroller	External	✓	-
	_setAppraisalOracle	External	✓	-

Contract Flow

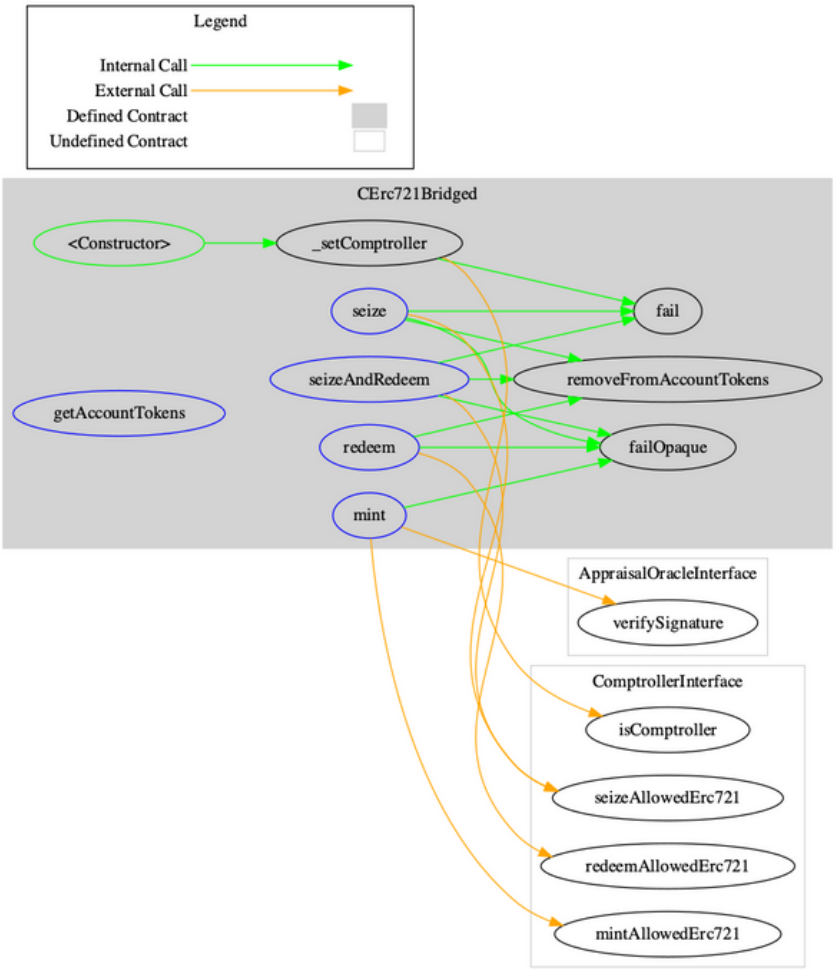
AppraisalOracle



CErc20



CErc721Bridged



Comptroller

Summary

MetaLend is a novel addition to the traditional compound protocol. It introduces NFTs for borrowing and supplying assets. Currently, there are not many technical reports and whitepapers in the industry regarding this methodology. Hence, there are some challenging decisions that should be taken about the way that the NFTs will interact with the compound protocol. We have identified some key architecture decisions and suggested potential solutions.

Disclaimer

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment.

Coinscope team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed.

The Coinscope team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Coinscope receive a payment to manipulate those results or change the awarding badge that we will be adding in our website.

Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token.

The Coinscope team disclaims any liability for the resulting losses.

About Coinscope

Coinscope is the leading early coin listing, voting and auditing authority firm. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.

Coinscope is aiming to make crypto discoverable and efficient globally. It provides all the essential tools to assist users draw their own conclusions.

The Coinscope.co team

https://www.coinscope.co