Idle Finance

Consensys Diligence

Date	June 2021
Auditors	Shayan Eskandari, Nicholas Ward

1 Executive Summary

This report presents the results of our engagement with **Idle Finance** to review **Idle Dynamic Tranches**.

The review was conducted by **Nicholas Ward** and **Shayan Eskandari** over the course of 20 person-days in June & July 2021.

2 Scope

The original review of 10 person-days began with commit hash `d94ee7194e8cb17db13b16c338f3e780b62f5435` and incorporated commit hash `26d190539dec83c603edc9b5a903ce9b29b33a07`, which provided a fix for an issue discovered independently by the development team. The complete list of files in scope for the initial review can be found in the Appendix. Due to the limited time available for the review, this was a best effort code review and does not cover the full extent of the codebase.

Note that many of the contracts in the scope call functions in the `idleToken` contracts which are out of the scope of the audit. We briefly reviewed the implementation of the Idle governance token to better understand the system.

Due to complexity of many of the calls (e.g. `harvest()`, `flashloan()`, etc) and the integration of many external DeFi projects(e.g. Compound, Uniswap, etc) the validity of the calculations and the code flow using purely manual review is close to impossible. We suggest an extensive integration and fuzzing test suites as an addition to the findings and recommendations in this report.

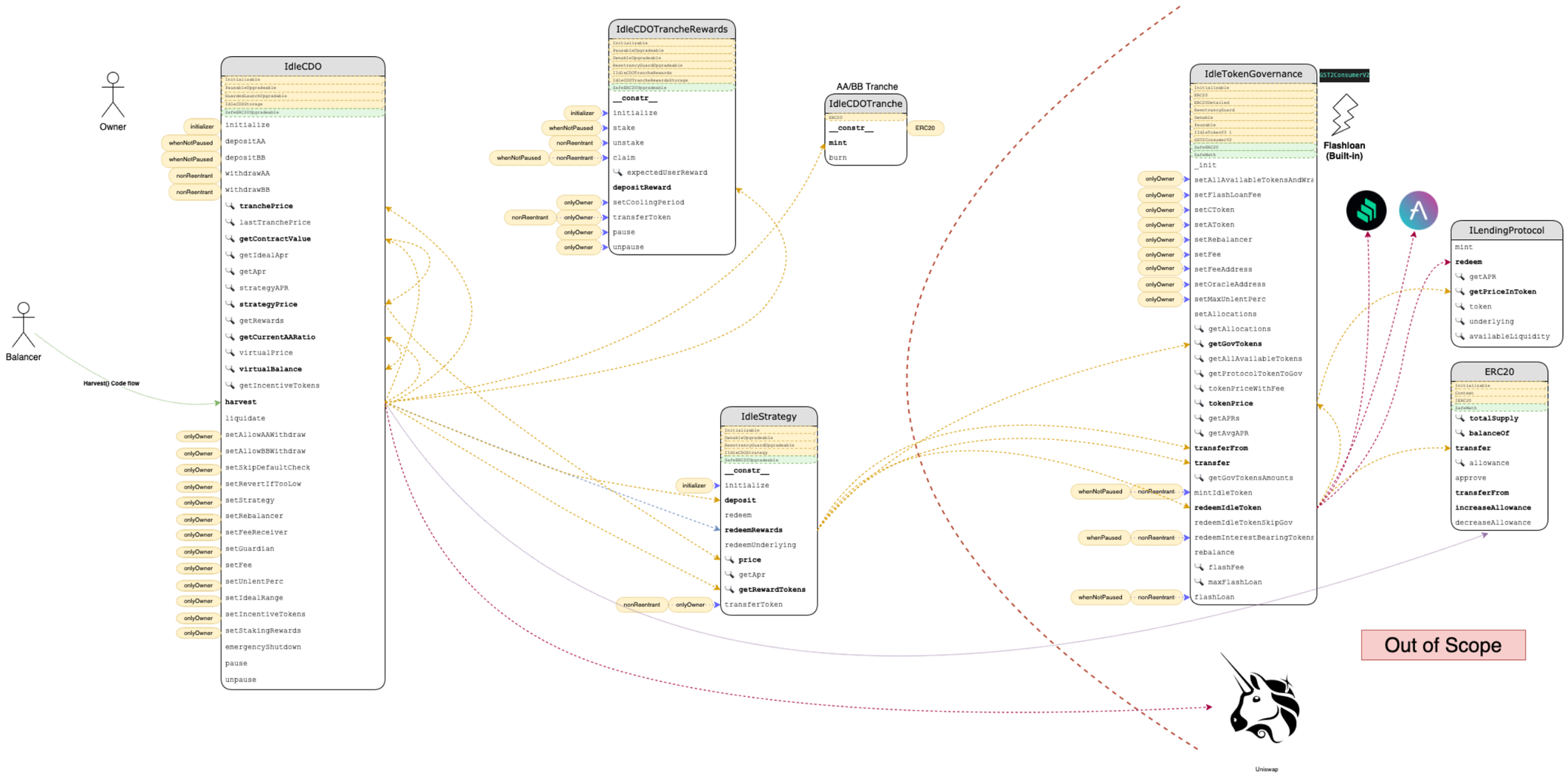
A second 10 person-day review focused largely on the `IdleCDO` contract. This review began with commit `1cf2b76bcda56807a35162ef4a65bcba0b6250e0` and incorporated commit `ff0b69380828657f16df8683c35703b325a6b656`.

A Mythx analysis report can be viewed here.

3 System Overview

Idle finance is comprised of many modules surrounding the main contract `IdleCDO.sol`. This includes ERC20 contracts for the Tranches and the interfaces for reward tokens, as well as Strategy contracts which will interact with different lending protocols. Many of the core functionality calls into `IdleTokenGovernance.sol` implementation, such as pricing and Idle token functionality.

The following is an overview of the Idle Finance contracts. The `harvest()` method was used to illustrate some of the contract interactions involved.



System Overview

Contracts are depicted as boxes. Public reachable interface methods are outlined as rows in the box. The 🔍 icon indicates that a method is declared as non-state-changing (view/pure) while other methods may change state. A yellow dashed row at the top of the contract shows inherited contracts. A green dashed row at the top of the contract indicates that the contract is used in a *usingFor* declaration. Modifiers used as ACL are connected as yellow bubbles in front of methods.

4 Recommendations

4.1 Prevent initialization of implementation contracts

Resolution

The development team addressed this recommendation in commit `f389c0a0d1ce0f245a8b82ac36767bfc0771a149`.

Description

The contracts `IdleCDO`, `IdleCDOTrancheRewards`, and `IdleStrategy` all inherit from OpenZeppelin's `Initializable` contract and are intended to be used via a delegatecall proxy. In general, implementation contracts should always be initialized on deployment or otherwise prevented from initialization by a malicious actor.

Recommendation

Add a constructor to every `Initializable` contract that sets `_initialized = true`. This constructor will be executed on deployment of implementation contracts, but will still allow proxy contracts that delegatecall to the implementation to use the `initialize()` method as before.

4.2 Clearly communicate admin capabilities

Description

With the exception of the internally-deployed `IdleCDOTranche` contracts, all of the contracts reviewed are intended to be used via delegatecall proxy. This allows the contract `owner` to upgrade to an arbitrary implementation at any time, potentially even front-running user calls to a contract with a change to the semantics of particular methods.

Additionally, the `IdleCDO` contract includes a function `transferTokens()` that allows the contract `owner` to withdraw any and all tokens held by the contract to the `governanceRecoveryFund` address.

code/contracts/GuardedLaunchUpgradable.sol:L54-L59

```
/// @notice Emergency method, tokens gets transferred to the governanceRecoveryFund address
/// @param _token address of the token to transfer
/// @param _value amount to transfer
function transferToken(address _token, uint256 _value) external onlyOwner nonReentrant {
  IERC20Upgradeable(_token).safeTransfer(governanceRecoveryFund, _value);
}
```

Further more, functionalities that can halt the system such as `emergencyShutdown` and Pausable contracts should be mentioned in a visible disclaimer.

Recommendation

Clearly communicate to users the trust model of the system, and use timelocked multisig contracts for controlling admin privileges.

4.3 Documentation for formulas and price/reward calculations

Resolution

The development team added related inline documentation in commits `e36b08e25fddb1195d774969e5c1da1d04507ea3` and `0cbe4fdb3bb17ba48dd42d0cd2cead4aaf9a7466`.

Description

Many of the calculations in the codebase uses nested function calls that are hard to read and verify. It is recommended to use inline documentation to indicate what the final formula should represent.

The complexity of these formulas reduces readability and maintainability of the code base. Other than proper documentation, It is critical to have full tests for these formulas (including happy path, edge cases, possible emergency functionalities, etc).

Examples

code/contracts/IdleCDO.sol:L145-L146

```
  return (_contractTokenBalance(_strategyToken) * strategyPrice() / (10**(strategyTokenDecimals))) + _contractTokenBalance(token);
}
```

code/contracts/IdleCDO.sol:L222-L224

```
function virtualBalance(address _tranche) public view returns (uint256) {
  return IdleCDOTranche(_tranche).totalSupply() * virtualPrice(_tranche) / ONE_TRANCHE_TOKEN;
}
```

5 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

5.1 Actors

The relevant actors are listed below with their respective abilities:

- **Owner**
  - (Deployer) upgrade all smart contracts (`GuardedLaunchUpgradable`: `SafeERC20Upgradeable`, `ReentrancyGuardUpgradeable`, `OwnableUpgradeable`)
  - Set flags to allow withdrawals for AA/BB tranches
  - Set flags to allow `setSkipDefaultCheck` and `setRevertIfTooLow` to change the system checks for default strategy and enable the check if redeemed amount during liquidations is enough

    - Can set strategy contracts to update the strategy used and potentially changing the lending protocol used
    - Change the *guardian*, *rebalancer*, *fee reciever* addresses
    - Change the fee, and unlent percentages
    - Change the Ideal range defining the range for AA/BB rewards
    - Change the incentive tokens used in the system
    - Change the tranche Rewards contract addresses
    - Can call `emergencyShutdown()` to pause deposits and redeems for all classes of tranches
    - Can pause and unpause the contracts
    - Can call `harvest()` to lend user funds in the lending provider through the IIdleCDOStrategy and update tranches incentives
    - Can call `liquidate()` to redeem underlyings from the lending provider
    - Transfer all "left over" tokens in the contracts (Strategy, TrencheRewards) to any address
    - Can set the `coolingPeriod` that a user needs to wait since his last stake before the unstake will be possible
- **Guardian**
    - Can call `emergencyShutdown()`
    - Can transfer funds to `governanceRecoveryFund` address
    - Can pause and unpause the contracts
- **Rebalancer**
    - Can call `harvest()`
    - Can call `liquidate()`
- **IdleCDO AA/BB Tranche Token holders**
    - Withdraw AA or BB tokens (if allowed by *owner*), which burns their tranche token and redeems their principal + interest
    - Can stake in the Tranche Reward contract. Note that if a user adds more stake later on, the `coolingPeriod` will restart
    - Can unstake if `coolingPeriod` is passed. If the contract is paused, "unstake" will skip the claim of the rewards
    - Can claim all the expected rewards
- `underlying token` holders
    - Deposit tokens and mint AA or BB Tranche Tokens
- feeReceiver
    - Receives fees through `_depositFees()` at harvest

6 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 `IdleCDO._deposit()` allows re-entrancy from hookable tokens. Medium

Resolution
The development team has addressed this concern in commit `5fbdc0506c94a172abbd4122276ed2bd489d1964`. This change has not been reviewed by the audit team.

Description

The function `IdleCDO._deposit()` updates the system's internal accounting and mints shares to the caller, then transfers the deposited funds from the user. Some token standards, such as ERC777, allow a callback to the source of the funds before the balances are updated in `transferFrom()`. This callback could be used to re-enter the protocol while already holding the minted tranche tokens and at a point where the system accounting reflects a receipt of funds that has not yet occurred.

While an attacker could not interact with `IdleCDO.withdraw()` within this callback because of the `_checkSameTx()` restriction, they would be able to interact with the rest of the protocol.

code/contracts/IdleCDO.sol:L230-L245

```
function _deposit(uint256 _amount, address _tranche) internal returns (uint256 _minted) {
  // check that we are not depositing more than the contract available limit
  _guarded(_amount);
  // set _lastCallerBlock hash
  _updateCallerBlock();
  // check if strategyPrice decreased
  _checkDefault();
  // interest accrued since last depositXX/withdrawXX/harvest is splitted between AA and BB
  // according to trancheAPRSplitRatio. NAVs of AA and BB are updated and tranche
  // prices adjusted accordingly
  _updateAccounting();
  // mint tranche tokens according to the current tranche price
  _minted = _mintShares(_amount, msg.sender, _tranche);
  // get underlyings from sender
  IERC20Detailed(token).safeTransferFrom(msg.sender, address(this), _amount);
}
```

Recommendation

Move the `transferFrom()` action in `_deposit()` to immediately after `_updateCallerBlock()`.

6.2 `IdleCDO.virtualPrice()` and `_updatePrices()` yield different prices in a number of cases Medium

Resolution

The development team implemented a new version of both functions using a third method, `virtualPricesAux()`, to perform the primary price calculation. Additionally, `_updatePrices()` was renamed to `_updateAccounting()`.

This change was incorporated in commit `ff0b69380828657f16df8683c35703b325a6b656`.

Description

The function `IdleCDO.virtualPrice()` is used to determine the current price of a tranche. Similarly, `IdleCDO._updatePrices()` is used to store the latest price of a tranche, as well as update other parts of the system accounting. There are a number of cases where the prices yielded by these two functions differ. While these are primarily corner cases that are not obviously exploitable in practice, potential violations of key accounting invariants should always be considered serious.

Additionally, the use of two separate implementations of the same calculation suggest the potential for more undiscovered discrepancies, possibly of higher consequence.

As an example, in `_updatePrices()` the precision loss from splitting the strategy returns favors BB tranche holders. In `virtualPrice()` both branches of the price calculation incur precision loss, favoring the `IdleCDO` contract itself.

`_updatePrices()`

code/contracts/IdleCDO.sol:L331-L341

```
if (BBTotSupply == 0) {
  // if there are no BB holders, all gain to AA
  AAGain = gain;
} else if (AATotSupply == 0) {
  // if there are no AA holders, all gain to BB
  BBGain = gain;
} else {
  // split the gain between AA and BB holders according to trancheAPRSplitRatio
  AAGain = gain * trancheAPRSplitRatio / FULL_ALLOC;
  BBGain = gain - AAGain;
}
```

`virtualPrice()`

code/contracts/IdleCDO.sol:L237-L245

```
if (_tranche == AATranche) {
  // calculate gain for AA tranche
  // trancheGain (AAGain) = gain * trancheAPRSplitRatio / FULL_ALLOC;
  trancheNAV = lastNAVAA + (gain * _trancheAPRSplitRatio / FULL_ALLOC);
} else {
  // calculate gain for BB tranche
  // trancheGain (BBGain) = gain * (FULL_ALLOC - trancheAPRSplitRatio) / FULL_ALLOC;
  trancheNAV = lastNAVBB + (gain * (FULL_ALLOC - _trancheAPRSplitRatio) / FULL_ALLOC);
}
```

Recommendation

Implement a single method that determines the current price for a tranche, and use this same implementation anywhere the price is needed.

6.3 `IdleCDO.harvest()` allows price manipulation in certain circumstances Medium

Resolution

The development team has addressed this concern in a pull request with a final commit hash of `5341a9391f9c42cadf26d72c9f804ca75a15f0fb`. This change has not been reviewed by the audit team.

Description

The function `IdleCDO.harvest()` uses Uniswap to liquidate rewards earned by the contract's strategy, then updates the relevant positions and internal accounting. This function can only be called by the contract `owner` or the designated `rebalancer` address, and it accepts an array which indicates the minimum buy amounts for the liquidation of each reward token.

The purpose of permissioning this method and specifying minimum buy amounts is to prevent a sandwiching attack from manipulating the reserves of the Uniswap pools and forcing the `IdleCDO` contract to incur loss due to price slippage.

However, this does not effectively prevent price manipulation in all cases. Because the contract sells it's entire balance of redeemed rewards for the specified minimum buy amount, this approach does not enforce a minimum *price* for the executed trades. If the balance of `IdleCDO` or the amount of claimable rewards increases between the submission of the `harvest()` transaction and its execution, it may be possible to perform a profitable sandwiching attack while still satisfying the required minimum buy amounts.

The viability of this exploit depends on how effectively an attacker can increase the amount of rewards tokens to be sold without incurring an offsetting loss. The strategy contracts used by `IdleCDO` are expected to vary widely in their implementations, and this manipulation could potentially be done either through direct interaction with the protocol or as part of a flashbots bundle containing a large position adjustment from an honest user.

code/contracts/IdleCDO.sol:L564-L565

```
function harvest(bool _skipRedeem, bool _skipIncentivesUpdate, bool[] calldata _skipReward, uint256[] calldata _minAmount) external {
  require(msg.sender == rebalancer || msg.sender == owner(), "IDLE:!AUTH");
```

code/contracts/IdleCDO.sol:L590-L599

```
// approve the uniswap router to spend our reward
IERC20Detailed(rewardToken).safeIncreaseAllowance(address(_uniRouter), _currentBalance);
// do the uniswap trade
_uniRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(
  _currentBalance,
  _minAmount[i],
  _path,
  address(this),
  block.timestamp + 1
);
```

Recommendation

Update `IdleCDO.harvest()` to enforce a minimum price rather than a minimum buy amount. One method of doing so would be taking an additional array parameter indicating the amount of each token to sell in exchange for the respective buy amount.

6.4 Prevent zero amount transfers/minting Minor

Resolution
The development team has addressed this concern in commit `a72747da8c0ca71274f3a1506c6faf724cf82dd2`. This change has not been reviewed by the audit team.

Description

Many of the functions in the system can be called with `amount = 0`. This is not a security issue, however a "defense in depth" approach in this and similar cases may prevent an undiscovered bug from being exploitable. Most of the functionalities that were reviewed in this audit won't create an exploitable state transition in these cases, however they will trigger a 0 token transfer or minting.

Examples

- depositAA()
- depositBB()
- stake()
- unstake()

Recommendation

Check and return early (or revert) on requests with zero amount.

6.5 Missing Sanity checks Minor

Resolution
The development team has addressed this concern in commit `a1d5dac0ad5f562d4c75bff99e770d92bcc2a72f`. This change has not been reviewed by the audit team.

Description

The implementation of `initialize()` functions are missing some sanity checks. The proper checks are implemented in some of the setter functions but missing in some others.

Examples

- Missing sanity check for `!= address(0)`

code/contracts/IdleCDO.sol:L54-L57

```
token = _guardedToken;
strategy = _strategy;
strategyToken = IIdleCDOStrategy(_strategy).strategyToken();
rebalancer = _rebalancer;
```

code/contracts/IdleCDO.sol:L84-L84

```
guardian = _owner;
```

code/contracts/IdleCDO.sol:L672-L673

```
address _currAAStaking = AAStaking;
address _currBBStaking = BBStaking;
```

code/contracts/IdleCDOTrancheRewards.sol:L50-L53

```
idleCDO = _idleCDO;
tranche = _trancheToken;
rewards = _rewards;
governanceRecoveryFund = _governanceRecoveryFund;
```

Recommendation

Add sanity checks before assigning system variables.

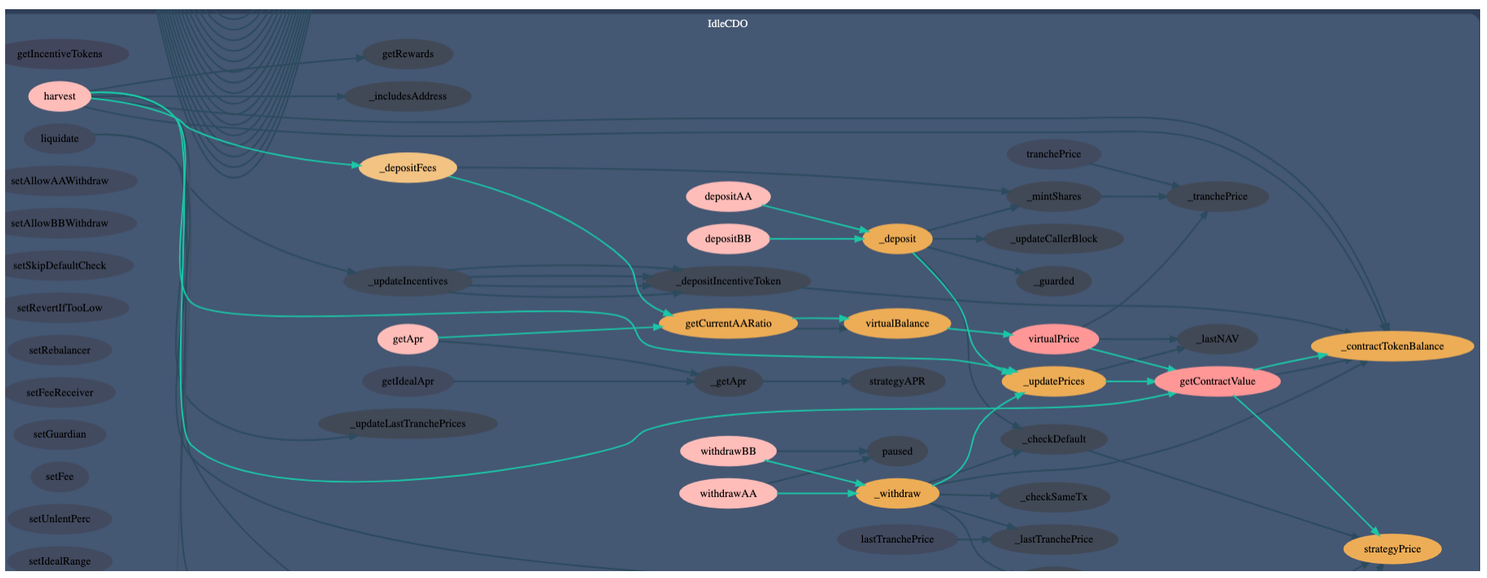
6.6 `IdleCDO.virtualPrice()` & `_updatePrices()` too complicated to verify Minor

Resolution
These methods were revisited in the continuation of the original review and more time was allotted to them than was possible previously. Some refactoring also occurred during that time (see 6.2). However, the development team elected to maintain the general approach used in these functions. The primary challenge in verifying their correctness remains, which is their heavy reliance on external interactions with contracts whose expected semantics are poorly defined.

Description

`IdleCDO.virtualPrice()` and `_updatePrices()` functions are used for many important functionality in the Idle system. They also have nested external calls to many other contracts (e.g. `IdleTokenGovernance`, `IdleCDOStrategy` and strategy token, `IdleCDOTranche` on both Tranche tokens, etc). This level of complexity for a vital function is not recommended and is considered dangerous implementation.

Examples



Recommendation

Consider refactoring the code to use less complicated logic and code flow.

Appendix 1 - Files in Scope

This audit covered the following files:

File Name	SHA-1 Hash
GuardedLaunchUpgradable.sol	5b2172e1874fa404f5dee86a4d0f05ed6a40287e
IdleCDO.sol	cf45736ad52997f1bd5795abc57cbc461975cbb2
IdleCDOStorage.sol	31c8ab77e4a0139080c4be4f6396ccf192d3ab7f
IdleCDOTranche.sol	4712775665bf393cb81e2950f8bc00658bcc38a5
IdleCDOTrancheRewards.sol	eaedc50169a341f1d533a3d3873d199ced67805b
IdleCDOTrancheRewardsStorage.sol	426f1f1705cdc494eae5d67e394bb8f0f223e286
interfaces/IIdleCDOStrategy.sol	bf623f6153c2a5c1889b24fcd630f2b0d618fa76
interfaces/IERC20Detailed.sol	b9faf29bb63e91308ee3152e0d2fb9bb9091fa5a

File Name	SHA-1 Hash
interfaces/IIdleCDOTrancheRewards.sol	97844b2e0df71e4f96f00eb4936ddd0cbfd00084

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.