Ondo RWA Internal Security Review

Auditors

Desmond Ho, Lead Security Researcher

Anurag Jain, Security Researcher

CarrotSmuggler, Associate Security Researcher

Report prepared by: Lucas Goiriz

March 27, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Ondo's mission is to make institutional-grade financial products and services available to everyone.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Ondo RWA Internal according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 19 days in total, Ondo Finance engaged with Spearbit to review the ondo-rwa-internal protocol. In this period of time a total of **25** issues were found.

Summary

Project Name	Ondo Finance
Repository	ondo-rwa-internal
Commit	1f9db1e9
Type of Project	DeFi, Stablecoin
Audit Timeline	Feb 25th to Mar 16th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	6	5	1
Gas Optimizations	7	6	1
Informational	11	8	3
Total	**25**	**20**	**5**

5 Findings

5.1 Medium Risk

5.1.1 Incorrect condition would cause DoS on subscription and redemption

Severity: Medium Risk

Context: OndoFees.sol#L301-L307

Description: User's `effective USD value` applicable for fees is determined via `_getEffectiveUSDValueAndUpdateUserFeeConfig` function. This function will currently fail under certain circumstances for `UserFeeMode.NO_FEE_AFTER_LIMIT` mode.

Proof of Concept:

1. Assume that for User `A`, current `userFeeConfig` is:

```
limitVolume : 10
currentVolume : 4
userFeeMode : UserFeeMode.NO_FEE_AFTER_LIMIT
```

2. User `A` now subscribes for same asset with deposit amount `7` (say post fees). This will cause net volume to go above `limitVolume` ($4 + 7 > 10$), so fee should only be taken on remaining limit which is `10 - 4 = 6`.
3. `effectiveUSDValue` is calculated as `4 - 10` instead of `10 - 4` which causes underflow error.

```
if (
  userFeeConfig.currentVolume + usdValue > userFeeConfig.limitVolume
) {
  effectiveUSDValue =
    userFeeConfig.currentVolume -
    userFeeConfig.limitVolume;
}
```

Coded Proof of Concept: Add this test in `forge-tests\xManager\OndoFees.t.sol`:

```
function test_POC() public {
    (, OndoFees.UserFeeConfig memory userFeeConfig) = ondoFees
      .getActiveFeeConfig(flatFeeRWAToken, address(USDC), aliceID);
    uint256 userVolumeWindow = 100;
    ondoFees.setUserFee(
      flatFeeRWAToken,
      aliceID,
      userFeeConfig.currentVolume,
      OndoFees.UserFeeConfig(
        OndoFees.FeeConfig(true, 10e18, 0),
        OndoFees.UserFeeMode.NO_FEE_AFTER_LIMIT,
        50e18, // limitVolume
        0, // currentVolume
        0, // lastReset
        userVolumeWindow // volumeWindow, seconds
      )
    );

    vm.warp(userVolumeWindow + 1);

    ondoFees.getAndUpdateFee(
      flatFeeRWAToken,
      address(USDC),
      aliceID,
      49e18
    );

    vm.expectRevert();
    ondoFees.getAndUpdateFee(
        flatFeeRWAToken,
        address(USDC),
        aliceID,
        2e18
      );

  }
```

Impact: This causes both subscription and redemption for `rwaToken` to fail.

Recommendation: Change `NO_FEE_AFTER_LIMIT` condition to:

```
  else if (userFeeConfig.userFeeMode == UserFeeMode.NO_FEE_AFTER_LIMIT) {
    // ...
    if (
      userFeeConfig.currentVolume + usdValue > userFeeConfig.limitVolume
    ) {
-       effectiveUSDValue = userFeeConfig.currentVolume - userFeeConfig.limitVolume;
+           effectiveUSDValue = userFeeConfig.limitVolume - userFeeConfig.currentVolume;
    } else {
        // ...
```

Ondo Finance: Fixed in commit 1b072be6.

Spearbit: Fix verified.

5.2 Low Risk

5.2.1 Precision loss in `rOUSG` share calculation allows subscriptions to exceed slippage tolerance

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In `subscribeRebasingOUSG()`, the minimum amount check is performed before several decimal conversions and share calculations, which can lead to precision loss. The actual rOUSG amount received may be less than the specified `minimumRwaReceived` due to rounding in `getSharesByROUSG()` and the OUSG to rOUSG share conversion.

Impact: Users may receive less rOUSG than their specified minimum, effectively bypassing their slippage protection.

Proof of Concept:

```
function test_subscribeRebasingOUSGMRealPrices() public {
    uint256 aliceDepositAmount = 100_000e6;
    deal(address(USDC), address(alice), aliceDepositAmount);

    vm.startPrank(OUSG_GUARDIAN);
    // use live price
    ousgOracleWrapper.setRwaOracle(0x0502c5ae08E7CD64fe1AEDA7D6e229413eCC6abe);
    // set ROUSG oracle to live price
    rOUSGToken.setOracle(0x0502c5ae08E7CD64fe1AEDA7D6e229413eCC6abe);

    vm.startPrank(alice);
    USDC.approve(address(ousgXManager), aliceDepositAmount);
    uint256 rousgAmountOut = ousgXManager.subscribeRebasingOUSG(
      address(USDC),
      aliceDepositAmount,
      1000e18
    );

    // assert minimum amount received
    assertGe(1000e18, rousgAmountOut); // this will revert
}
// The POC reverts with a failing assertion. `[FAIL: assertion failed: 100000000000000000000 <
↪  99999999999999999999903]`
```

Recommendation: Consider adding a direct slippage check at the end against `minimumRwaReceived` (which should be renamed to `minimumROUSGReceived` for clarity).

```
if (rousgAmountOut < minimumRwaReceived) revert RwaReceiveAmountTooSmall();
```

Ondo Finance: Fixed in commit bc1c3b14.

Spearbit: Fix verified.

5.2.2 Unconditional rounding up pulls in 1 wei more for divisible USDS amounts

Severity: Low Risk

Context: PSMSource.sol#L158-L159

Description: The conversion from USDS to USDC unconditionally adds 1 after division. This means that even when the USDS amount is perfectly divisible by the conversion rate, the contract will still pull 1 extra wei of USDC, leading to dust amounts being left in the contract.

This is especially important for when `BuidlUSDCSource` is the `USDCSource`. Should there be a `BUIDL` redemption required: `_redeemBUIDL(requestedWithdrawAmount - usdcBalanceBefore)`, the unconditional round-up will cause the redemption to fail because there isn't sufficient `BUIDL` for the additional requested USDC wei.

Recommendation: Use a ceiling division formula that only rounds up when necessary.

```
// ref: https://github.com/balancer/balancer-v2-monorepo/blob/master/pkg/solidity-utils/contracts/math/
↪  FixedPoint.sol#L61-L64
// divUp(x, y) := (x + y - 1) / y
uint256 usdcAmountNeeded = (usdsAmountNeeded +
   usdcToUsdsDecimalConversionRate - 1) / usdcToUsdsDecimalConversionRate;
```

Ondo Finance: Fixed in commit c0210e5e.

Spearbit: Fix verified.

5.2.3 Setting limits should revive `capacityUsed` accordingly

Severity: Low Risk

Context: OndoRateLimiter.sol#L367-L379

Description: Currently, if `GlobalSubscriptionLimit/GlobalRedemptionLimit/UserSubscriptionRateLimit/UserRedemptio` is set by `CONFIGURER_ROLE` then `capacityUsed` is reset to `0`.

This can cause issues since User might have already exhausted their limits and this call revive full limit without waiting for `window` period.

Proof of Concept:

1. Attacker frontrun `setGlobalSubscriptionLimit` call and subscribe fully to current limit.
2. `setGlobalSubscriptionLimit` call executes and sets `newLimit`. At same time it also sets `capacityUsed` as `0`.
3. Attacker can again subscribe with the `newLimit` instantly without need to wait.

Recommendation: New `capacityUsed` can be set as per `Current capacity used percentage`. Example:

1. If old limit was `100` and `50%` capacity was used.
2. Now if limit is changed to `200` then new `capacityUsed` can be set to `100 (50%)`.

Ondo Finance: Acknowledged. If we are ever concerned about the limits being abused during a change, we can pause the manager contracts prior to setting the new limits. Regardless, when setting new rate limits we can always be aware that this may happen and simply be prepared to process more funds as opposed to adding contract complexity.

Spearbit: Acknowledged.

5.2.4 `availableToWithdraw` does not check paused state

Severity: Low Risk

Context: BuidlUSDCSource.sol#L202-L210, PSMSource.sol#L186-L201

Description: Due to how the `OndoTokenRouter` contract is set up, it expects the result of the `availableToWithdraw` call to be available for withdrawal from all the token sources. If the `availableToWithdraw` function returns a non-zero value but this amount is not actually obtainable by calling the `tokenSource.withdrawToken` function, it can lead to reverts or broken accounting.

```
uint256 amountAvailable = tokenSource.availableToWithdraw(outputToken);

uint256 withdrawAmount = amountAvailable > requestedWithdrawAmount
  ? requestedWithdrawAmount
  : amountAvailable;

// INVARIANT - `TokenSource.withdrawToken` will always withdraw the amount requested
// or revert.
tokenSource.withdrawToken(outputToken, withdrawAmount);
```

Some of the token source contracts are pausable. In case they are paused, the amount reported should be 0, so that the router does not try to extract tokens from it.

The `BuidlUSDCSource` contract reports the available liquidity in the BUIDL settlement contract but doesn't check if the contracts are paused, and thus, if this amount is actually obtainable. If the BUIDL redemption is paused, the `withdrawToken` calls to this token source contract can revert.

Similarly, the `PSMSource` contract uses the USDS PSM module which implements a `tin` parameter which assigns the paused state. This can be verified here at this address:

- `0xA188EEC8F81263234dA3622A406892F3D630f98c`.

Calls to this contract can also revert when it is paused.

Proof of Concept: Assume the `BuidlUSDCSource` has 100 USDC tokens, 900 BUIDL tokens and the BUIDL settlement contract has enough liquidity but is paused. The `availableToWithdraw` function will report 1000 USDC. However the contract can only dispense up to 100 USDC. Any more and the `redeem` function will be called on the BUIDL redeemer, and the transaction will revert since it is paused.

Recommendation: Check the paused state of underlying contracts. If paused, only the current USDC balance in the token source contract is available and any redemptions/swaps are offline.

Ondo Finance: Fixed in commit a876a50b.

Spearbit: Fix verified.

5.2.5 `PSMSource.sol:availableToWithdraw` does not check available USDS liquidity

Severity: Low Risk

Context: PSMSource.sol#L187

Description: The USDS peg stabilization module can be used to swap USDS for USDC tokens and vice versa at 1:1 ratios. The current USDS PSM actually just uses the older DAI PSM module underneath it. Thus, during a USDC $\rightarrow$ USDS swap, the following steps take place:

1. USDC is transferred from the user to the USDS PSM contract at `0xA188EEC8F81263234dA3622A406892F3D630f98c`.
2. The USDS PSM contract uses the USDC to buy DAI from the DAI PSM contract at `0xf6e72Db5454dd049d0788e411b06CfAF16853042`.
3. The DAI is then migrated to USDS.

Thus, this flow only works as long as there is enough DAI in the DAI PSM module. The `availableToWithdraw` in this contract however reports the amount of USDS available as the USDC balance of the contract itself. This can be incorrect in case not enough DAI is available in the PSM contracts. The `availableToWithdraw` reports a value higher than is actually available, which can lead to reverts in the Ondo token router contract since it always expects the reported available balance to be withdrawable.

Recommendation: When reporting the available amount of USDS tokens, the amount of DAI available in the DAI PSM contract needs to be considered.

Ondo Finance: Fixed in commit 89a5f6ad.

Spearbit: Fix verified.

5.2.6 `BuidlUSDCSource` can have fewer BUIDL tokens than the minimum redeem criteria

Severity: Low Risk

Context: BuidlUSDCSource.sol#L168, BuidlUSDCSource.sol#L196-L198

Description: The `BuidlUSDCSource` contract reports the amount of USDC tokens it can provide through its `availableToWithdraw` function, and redeems the BUIDL tokens it holds in order to get this USDC amount. Due to how the Ondo token router is set up, the amount reported by `availableToWithdraw` must always be available for withdrawal from the token contract. However this can get violated if the contract is holding less BUIDL tokens than

the minimum redeem limit. The `availableToWithdraw` simply reports the current BUIDL balance of the contract, expecting it all to be available for withdrawal.

```
uint256 availableRedeemerLiquidity = ISettlement(buidlRedeemer.settlement())
      .availableLiquidity();

uint256 availableBuidlRedemptionTotal = currentBuidlBalance >
      availableRedeemerLiquidity
      ? availableRedeemerLiquidity
      : currentBuidlBalance;

return currentUSDCBalance + availableBuidlRedemptionTotal;
```

But if the contract holds less than the minimum BUIDL redemption amount, none of its BUIDL tokens are actually redeemable. In that case, the contract will be over-reporting the amount of USDC liquidity it can provide, and will thus revert if the reported amount is attempted to be withdrawn from it.

Proof of Concept: Assume the contract has 0 USDC and 190k BUIDL tokens. Assume the minimum redemption limit is 100k BUIDL. When a user calls to withdraw 90k USDC tokens, the router calls this contract to redeem 100k BUIDL tokens for 100k USDC tokens. After paying out the user the contract is left with 10k USDC and 90k BUIDL tokens.

Now, the `availableToWithdraw` reports that it can provide 10k+90k = 100k of liquidity. But if this 100k liquidity is to be withdrawn, the contract needs to redeem the 90k BUIDL balance, which is impossible since it falls below the minimum redemption limit. Thus the contract can only provide up to 10k USDC.

Recommendation: In the `availableToWithdraw` function, add the following:

```
currentBuidlBalance = currentBuidlBalance >= minBUIDLRedeemAmount ? currentBuidlBalance : 0;
```

This will make sure the BUIDL balance is only considered if it is actually redeemable.

Ondo Finance: Fixed in commit fa8c22c3.

Spearbit: Fix verified.

5.3 Gas Optimization

5.3.1 Redundant allowance checks

Severity: Gas Optimization

Context: BaseRWAManager.sol#L170-L173, OUSG_InstantManager.sol#L240, OUSG_InstantManager.sol#L267, BasicRecipient.sol#L64-L67, UsdsPSMRecipient.sol#L103-L106

Description: Checking the token allowance isn't necessary prior to the invocation of `safeTransferFrom()`, because the transfer should revert if there's insufficient allowance.

Recommendation: The allowance checks can be removed.

Ondo Finance: Fixed in commit 5319d05b.

Spearbit: Fix verified.

5.3.2 Redundant `InsufficientPsmUsdc` check

Severity: Gas Optimization

Context: UsdsPSMRecipient.sol#L112-L113

Description: `buyGem()` will revert if there's insufficient USDC in `psm.pocket()`, so the check isn't necessary, though it will provide a clearer revert reason.

Proof of Concept:

```
function test_deposit_failInsufficientUsdcWithRealPsm() public {
    uint256 usdcAvailableBalance = USDC.balanceOf(address(PSM.pocket()));
    // increase by 1 to exceed available balance
    uint256 usdsDepositAmount = (usdcAvailableBalance + 1) * 1e12;
    deal(address(USDS), depositor, usdsDepositAmount);

    // Approve recipient to transfer tokens
    vm.startPrank(depositor);
    USDS.approve(address(recipient), usdsDepositAmount);
    recipient.depositToken(address(USDS), usdsDepositAmount);
    vm.stopPrank();
}
```

Recommendation: Consider removing this check.

Ondo Finance: Acknowledged. Gas optimizations generally aren't a concern for our contracts outside of low-hanging fruit. Due to the value of the transactions through these contracts, it is much better to optimize for cleaner code than minor gas savings, and in this case the better error message is more beneficial.

Spearbit: Acknowledged.

5.3.3 Redundant decrement in `SusdsSource.availableToWithdraw()`

Severity: Gas Optimization

Context: SusdsSource.sol#L115-L120

Description: The subtraction by 1 is redundant because `susds.convertToAssets()` rounds down.

Proof of Concept:

```
Fuzz testing with some practical values of `susds` found no issues with using the output of
↪  `availableToWithdraw()`.

function test_fuzzWithdrawMaxAmount(uint256 susdsAmount) public {
    // bound sudsAmount to be between 1 and 1_000_000e18
    susdsAmount = bound(susdsAmount, 1, 1_000_000e18);
    // deal amount to withdrawAddress
    deal(address(SUSDS), withdrawAddress, susdsAmount);
    vm.prank(withdrawAddress);
    SUSDS.approve(address(source), susdsAmount);

    vm.startPrank(withdrawer);
    source.withdrawToken(address(USDS), source.availableToWithdraw(address(USDS)));
}
```

Recommendation:

```
- uint256 usdsAvailable = susds.convertToAssets(susdsAvailable);
- return usdsAvailable > 0 ? usdsAvailable - 1 : 0;
+ return susds.convertToAssets(susdsAvailable);
```

Ondo Finance: Fixed in commit 406a9e65.

Spearbit: Fix verified.

5.3.4 Redundant `USD_NORMALIZER` normalization

Severity: Gas Optimization

Context: BaseRWAManager.sol#L206-L209

Description: The RWA amount calculation includes a multiplication and division by `USD_NORMALIZER` that cancel each other out.

Recommendation:

```
rwaAmountOut = (depositUSDValue - fee) * RWA_NORMALIZER) / _getRwaPrice();
```

Ondo Finance: Fixed in commit 83b35ccb.

Spearbit: Fix verified.

5.3.5 Redundant `uint48` casting of `window`

Severity: Gas Optimization

Context: OndoRateLimiter.sol#L378, OndoRateLimiter.sol#L406

Description: The `window` parameter is already declared as `uint48` in function signatures, but is redundantly cast to `uint48` when assigning to the struct.

Recommendation: Remove the redundant type casting since the parameter is already the correct type:

```
- window: uint48(window)
+ window: window
```

Ondo Finance: Fixed in commit d666eb36.

Spearbit: Fix verified.

5.3.6 First conditional volume check can include equality case

Severity: Gas Optimization

Context: OndoFees.sol#L285-L296

Description: In the case where `userFeeConfig.currentVolume == userFeeConfig.limitVolume`, the `effectiveUSDValue` would simply be `usdValue`, which is the first case, so the direct assignment would be desired.

Recommendation:

```
- if (userFeeConfig.currentVolume > userFeeConfig.limitVolume) {
+ if (userFeeConfig.currentVolume >= userFeeConfig.limitVolume) {
```

Ondo Finance: Fixed in commit cbb26f99.

Spearbit: Fix verified.

5.3.7 `currentPrice` can be directly assigned to `priceStored`

Severity: Gas Optimization

Context: ContinuousPriceOracle.sol#L243.

Description: Since the current price has been synced via `_syncRateAndPriceStoredToNow()`, `currentPrice` can simply read from the `priceStored` variable.

Recommendation:

```
- uint256 currentPrice = getPrice();
+ uint256 currentPrice = priceStored;
```

Ondo Finance: Fixed in commit 90f8c4c3.

Spearbit: Fix verified.

5.4 Informational

5.4.1 Comment and Variable Improvements

Severity: Informational

Context: OndoIDRegistryFactory.sol#L81, AdminSubscriptionChecker.sol#L45, OUSG_InstantManager.sol#L143, OUSG_InstantManager.sol#L151, OUSG_InstantManager.sol#L255, PauseManager.sol#L88, OndoTokenRouter.sol#L211, OndoTokenRouter.sol#L311

Description: The referenced lines have typos or incorrect comments or variables that can be modified for better clarity.

Recommendation:

```
- 2) Will transfer ownership of the proxyAdmin to guardian address.
+ 2) Will transfer ownership of the proxyAdmin to `admin` address.

- demoninated
+ denominated

- minimumRwaReceived
+ minimumRousgReceived
- * @param  minimumRwaReceived Minimum amount of RWA to receive, in decimals of rOUSG
+ * @param  minimumRousgReceived Minimum amount of rOUSG to receive

- minimumOusgAmonut
+ minimumOusgAmount

- rwaAmount
+ rousgAmount

- Constuctor
+ Constructor

- avaialable
+ available

-  * @param _minimumTokenPrice The minimum price for the token, denoted in USD with 18 decimals
+  * @param _minimumTokenPrice The minimum price for the token, denoted in USD with oracle decimals
```

Ondo Finance: Fixed in commit 2bc2ab31.

Spearbit: Fix verified.

5.4.2 Consider allowing current used capacity to be carried over

Severity: Informational

Context: OndoRateLimiter.sol#L375, OndoRateLimiter.sol#L403, OndoRateLimiter.sol#L486

Description: `capacityUsed` is always reset whenever `setGlobalRedemptionLimit()`, `setUserSubscriptionRateLimit()` or `setUserRedemptionRateLimit()` is called. There could be a use-case for the limit and / or window to be changed whilst leaving the current capacity used intact. As such, it would be great for a boolean flag to be passed to indicate whether or not to reset the current capacity.

Recommendation: Here's an example for `setGlobalRedemptionLimit()`, where a boolean `rolloverCapacity` is added.

```
uint256 globalCurrentCapacityUsed;

if (rolloverCapacity) {
   RateLimit memory globalRl = globalSubscriptionLimits[rwaToken];
   (globalCurrentCapacityUsed, ) = _calculateDecay(
      globalRl.capacityUsed,
      globalRl.lastUpdated,
      globalRl.limit,
      globalRl.window
   );
}

globalSubscriptionLimits[rwaToken] = RateLimit({
   capacityUsed: globalCurrentCapacityUsed;
   // ...
});
```

Ondo Finance: Acknowledged. Solid suggestion, but think it adds more complexity to the code than it's worth - we'd also have to ensure that the capacity being rolled over is less than the new limit to avoid unexpected behavior. If there's ever a case where operationally we can't handle an effective higher limit due to the capacity being nulled out (highly highly unlikely), we could instead temporarily pause the contracts.

Spearbit: Acknowledged.

5.4.3 Missing interface inheritance of `PauseManager`

Severity: Informational

Context: PauseManager.sol#L28

Description: The `PauseManager` contract imports but does not inherit from `IPauseManager`. Additionally, the interface contains outdated function signatures that don't match the actual implementation (e.g., `pauseSpecificContract()` vs `pauseContract()`).

Recommendation: `IPauseManager` should be updated, and have `PauseManager` inherit it.

Ondo Finance: Fixed in commit 252b69f1.

Spearbit: Fix verified.

5.4.4 `FEE_PRECISION` is of inconsistent precision

Severity: Informational

Context: OndoFees.sol#L49-L50

Description: `OndoFees` uses `FEE_PRECISION = 1e18` for basis point calculations, while other contracts in the system use the standard `1e4` (10_000) for basis points.

Recommendation: Align with the standard basis point precision used throughout the system:

```
- uint256 public constant FEE_PRECISION = 1e18;
+ uint256 public constant FEE_PRECISION = 1e4;
```

Ondo Finance: Fixed in commit 59866be7.

Spearbit: Fix verified.

5.4.5 Enforce both `minimumTokenPrice` and `oracle` are set

Severity: Informational

Context: OndoTokenRouter.sol#L372-L373

Description: If a `minimumTokenPrice[token]` is set, then it must be always enforced. Although it was observed that if `address(tokenToAggregatorV3Oracle[token].oracle` is unset then `minimumTokenPrice` is not respected and operation continues without minimum price.

Proof of Concept:

1. Observe the `_assertTokenMinimumPrice` function.

```
function _assertTokenMinimumPrice(address token) internal view {
    // If the minimum price and/or oracle is not set, then we don't need to check the minimum price.
    if (minimumTokenPrice[token] == 0) return;
    if (address(tokenToAggregatorV3Oracle[token].oracle) == address(0)) return;
    // ...
}
```

2. Let's say `minimumTokenPrice[token]` was set as `X` but `address(tokenToAggregatorV3Oracle[token].oracle) = address(0)`.
3. In this case, `_assertTokenMinimumPrice` will return without throwing any error which means minimum price check was not performed.

Recommendation:

1. If `minimumTokenPrice[token]` is set then revert if `address(tokenToAggregatorV3Oracle[token].oracle` is unset.
2. Otherwise ensure that both `minimumTokenPrice[token]` and `address(tokenToAggregatorV3Oracle[token].oracle` are set together.

Ondo Finance: Fixed in commit 5565854c.

Spearbit: Fix verified.

5.4.6 Max cap can be set on `feeConfig.bpsFee`

Severity: Informational

Context: OndoFees.sol#L538-L564

Description: There is currently on max cap set on `feeConfig.bpsFee` for all fee modes. If set to a a very high value then User could lose most of deposit funds towards fees.

Recommendation: Apply a max fee cap on `feeConfig.bpsFee` for all fees mode.

Ondo Finance: Fixed in commit 152eac3a.

Spearbit: Fix verified.

5.4.7 `ORACLE_SETTER_ROLE` can change prices even on Freeze situation

Severity: Informational

Context: ContinuousPriceOracle.sol#L336

Description: If due to some situation, `freezePrice` is called then `currentYearlyRate` is set to `0`.

```
function freezePrice() external onlyRole(PRICE_FREEZER_ROLE) {
    _syncRateAndPriceStoredToNow();
    currentYearlyRate = 0;
    // ...
  }
```

This does 2 things:

1. Prevents setter from changing rates using `setRateNow` or `setNextRate` since below check always fails.

```
if (
    rateChange > (MAX_RATE_CHANGE_BPS * currentYearlyRate) / BPS_PRECISION
) revert RateChangeTooLarge();

// Always fail since currentYearlyRate=0
```

2. But it seems that `ORACLE_SETTER_ROLE` can still change current price by using `setPriceNow` max upto `MAX_PRICE_CHANGE_BPS` which is currently set to 2%.
3. Ideally only Admin should be able to change rate or prices in case `freezePrice` has been called. In current code, changing rate can only be done by `Admin` but prices can changed by `ORACLE_SETTER_ROLE` as well.

Recommendation: Do not allow `ORACLE_SETTER_ROLE` to change prices if freezing period is ongoing (i.e. currentYearlyRate=0).

Ondo Finance: Acknowledged. Freeze price is mainly made for the case where the rate is set far too high and we need to stop accrual. Given this, adjusting the price within the set limits is fine after a freeze.

Spearbit: Acknowledged.

5.4.8 `defaultUserSubscriptionLimitConfigs` is only applicable for new Users

Severity: Informational

Context: OndoRateLimiter.sol#L430-L433

Description: Changes to `defaultUserSubscriptionLimitConfigs` will only be applicable to new users and old users limit wont be changed.

Proof of Concept:

1. Default user limit for `defaultUserSubscriptionLimitConfigs` is currently set to `10M`.
2. User `A` subscribes and since he is a new User, he gets limit of `10M`.

```
RateLimit storage userRl = transactionType == TransactionType.SUBSCRIPTION
    ? userSubscriptionLimits[rwaToken][userID]
    : userRedemptionLimits[rwaToken][userID];

    // If the user rate limit has not been set, instantiate it with the default configuration
    if (userRl.lastUpdated == 0)
      _instantiateUserRateLimits(transactionType, rwaToken, userID);
```

3. Limit for `defaultUserSubscriptionLimitConfigs` is changed to `5M`.

```
function setDefaultUserSubscriptionLimitConfig(
    address rwaToken,
    uint256 limit,
    uint48 window
  ) external onlyRole(CONFIGURER_ROLE) {
    if (rwaToken == address(0)) revert RWAAddressCantBeZero();

    defaultUserSubscriptionLimitConfigs[rwaToken] = RateLimitConfig({
      limit: limit,
      window: window
    });
    // ...
```

4. User `A` still enjoys limit of `10M` and revised change is only applicable to new users.

Note: Same issue holds for `defaultUserRedemptionLimitConfigs` which new redemption limit is only applied to new users.

Recommendation: One possible way could be to add `feeType` to say `RateLimit` which could have values as `DEFAULT` or `SPECIAL`. If `DEFAULT` then we can verify that current limit for user is same as default user limit, if not we can reset user limit to default limit. This would also require the `capacityUsed` to be reset as per new default limit.

Ondo Finance: Acknowledged. We will be constantly monitoring user states off chain, and it is known that operationally we will need to retroactively edit rate limits in the case of changing defaults.

Spearbit: Acknowledged.

5.4.9 Frontrunning can be used to disallow Fee Manager operations

Severity: Informational

Context: OndoFees.sol#L549-L554

Description: User can fail `FEE_MANAGER_ROLE` attempt to change User fee configs by frontrunning. Any simple volume change will cause `FEE_MANAGER_ROLE` call to fail.

Proof of Concept:

1. Fee Manager decides to stop giving special fees to a `userID`.
2. It calls `setUserFeeOverride` with `userFeeOverride[rwaToken][stablecoin][userID].feeConfig.active` as `false`.
3. User from impacted `userID` immediately frontrun by subscribing thus changing `currentVolume`.
4. Fee Manager call fails as volume has changed.

```
if (
  currentVolume !=
  userFeeOverride[rwaToken][stablecoin][userID].currentVolume
) revert CurrentVolumeIncorrect();
```

Note: Same situation applies for `setUserFeeConfigWindow` and `setUserFee` as well.

Recommendation: Important operations such as changing active status should not depend on `currentVolume` changes and should be applied irrespective of any volume changes. This can also be used to remove users who are frontrunning to avoid limit reduction.

Ondo Finance: Fixed in commit e7621a58.

Spearbit: Fix verified.

5.4.10 `withdrawToken` call should be skipped for 0 token withdrawals

Severity: Informational

Context: OndoTokenRouter.sol#L242

Description: The `_gatherTokensFromWithdrawalSources` function in the token router contract iterates over the various token sources, checks how much funds are available and then takes that amount out of those sources. The issue is that sometimes there can be 0 funds available in a source. In that case, the contact still tries to take out 0 tokens from that source.

```
uint256 amountAvailable = tokenSource.availableToWithdraw(outputToken); //@audit can be 0

uint256 withdrawAmount = amountAvailable > requestedWithdrawAmount
        ? requestedWithdrawAmount
        : amountAvailable;

// INVARIANT - `TokenSource.withdrawToken` will always withdraw the amount requested
// or revert.
tokenSource.withdrawToken(outputToken, withdrawAmount); //@audit calling withdrawToken with 0
```

The issue is that some of the token sources can revert on a zero amount withdrawal call.

1. If the BUIDL token redemption is paused, the BUIDL USDC source contract might report an available balance of 0. The `withdrawToken` call will then force a call to the redemption contract reverting the transaction.
2. If the USDS PSM contract is halted, then buy/sell calls to it even with 0 amounts will revert.

Recommendation: If the `withdrawAmount` is 0, skip the `tokenSource.withdrawToken` call.

Ondo Finance: Fixed in commit ca06d338.

Spearbit: Fix verified.

5.4.11 `ADMIN_SUBSCRIPTION_ROLE` can mint during pause

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: `ADMIN_SUBSCRIPTION_ROLE` can still mint token even when contract subscription is in paused state. This could become a problem if `ADMIN_SUBSCRIPTION_ROLE` is compromised. In that case, Attacker can use `ADMIN_SUBSCRIPTION_ROLE` to mint tokens via `adminSubscribe` function and sell them in external market (if any). Although impact is limited to `min(globalSubscriptionLimit,adminSubscriptionAllowance[admin])`.

Recommendation: Add `whenSubscribeNotPaused` to `adminSubscribe` function.

Ondo Finance: Fixed in commit 542e1abc.

Spearbit: Fix verified.