SPEARBIT

Ondo RWA Internal Security Review

Auditors

Desmond Ho, Lead Security Researcher

Anurag Jain, Security Researcher

CarrotSmuggler, Associate Security Researcher

Report prepared by: Lucas Goiriz

June 17, 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 7 days in total, Ondo Finance engaged with Spearbit to review the rwa-internal protocol. In this period of time a total of **7** issues were found.

Summary

Project Name	Ondo Finance
Repository	rwa-internal
Commit	88353254
Type of Project	DeFi, Stablecoin
Audit Timeline	May 28th to Jun 4th

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	3	2	1
Total	**7**	**6**	**1**

5 Findings

5.1 Medium Risk

5.1.1 `GMToken` isn't redeemable due to lacking `burn(uint256 amount)` function

Severity: Medium Risk

Context: GMToken.sol#L21, GMTokenManager.sol#L251-L257

Description: Redeeming GM tokens for RWA tokens will revert, because the redemption attempts to call `burn(quote.quantity)` on the `GMToken`, which it doesn't implement.

Proof of Concept:

```
diff --git a/forge-tests/globalMarkets/gmTokens/tokenDeployIntegration.t.sol
↪  b/forge-tests/globalMarkets/gmTokens/tokenDeployIntegration.t.sol
index b763df1..c4d485c 100644
--- a/forge-tests/globalMarkets/gmTokens/tokenDeployIntegration.t.sol
+++ b/forge-tests/globalMarkets/gmTokens/tokenDeployIntegration.t.sol
@@ -29,6 +29,7 @@ contract GMTokenFactoryTest_ETH is OUSG_InstantManager_BasicDeployment {
   OndoComplianceGMView public ondoComplianceGMView;
   TokenPauseManager public tokenPauseManager;
   GMTokenManager public gmTokenManager;
+  address public newGmToken;
   IssuanceHours public issuanceHours;
   onUSD public onusd;

@@ -152,7 +153,7 @@ contract GMTokenFactoryTest_ETH is OUSG_InstantManager_BasicDeployment {

   function test_deployAndMintNewGMToken() public {
     vm.prank(OUSG_GUARDIAN);
-    address newGmToken = gmTokenFactory.deployAndRegisterToken(
+    newGmToken = gmTokenFactory.deployAndRegisterToken(
       "Test GM Token",
       "TGT",
       OUSG_GUARDIAN
@@ -204,6 +205,41 @@ contract GMTokenFactoryTest_ETH is OUSG_InstantManager_BasicDeployment {
     assertEq(onusd.totalSupply(), onUsdSupply - depositAmount);
   }

+  function test_deployMintAndRedeemNewGMToken() public {
+    test_deployAndMintNewGMToken();
+
+    uint256 rwaAmount = 1e18;
+    uint256 expiration = block.timestamp + 1 hours;
+
+    // now do redemption
+    IGMTokenManager.Quote memory quote = IGMTokenManager.Quote({
+      chainId: block.chainid,
+      attestationId: 2,
+      userId: aliceID,
+      asset: address(newGmToken),
+      price: 150e18,
+      quantity: rwaAmount,
+      expiration: expiration,
+      side: IGMTokenManager.QuoteSide.SELL,
+      additionalData: ""
+    });
+
+    // Create the attestation signature
+    bytes memory signature = _createAttestation(attesterPrivateKey, quote);
+
+    // Approve manager to spend user's newGmToken
+    vm.startPrank(alice);
+    IERC20(newGmToken).approve(address(gmTokenManager), rwaAmount);
+
+
+    gmTokenManager.redeemWithAttestation(
+      quote,
+      signature,
+      address(onusd),
+      0
+    );
+  }
+
   function _createAttestation(
     uint256 signerPrivateKey,
     IGMTokenManager.Quote memory quote
```

Gets a generic revert because there isn't a function that matches the requested function selector when running `forge test --mt test_deployMintAndRedeemNewGMToken --rpc-url mainnet`. The test passes after adding:

```
function burn(uint256 amount) external {
   _burn(msg.sender, amount);
}
```

Recommendation: Have `GMToken` inherit `ERC20BurnableUpgradeable` (and `IRWALIke` interface), as it contains both the `burn(uint256 amount)` and `burnFrom(address from, uint256 amount)` functions. Also, as an optimisation, `burnFrom()` can be called instead of `safeTransferFrom()` and `burn()`.

```
IRWALike(quote.asset).burnFrom(_msgSender(), quote.quantity);
```

Ondo Finance: Fixed in PR 432.

Spearbit: Fix verified.

5.1.2 `QUOTE_TYPEHASH` is incorrectly defined

Severity: Medium Risk

Context: GMTokenManager.sol#L69, IGMTokenManager.sol#L48-L58

Description: The typehash is incorrect for enum `QuoteSide`, as it is converted to `uint8`. By running `forge eip712 ./contracts/globalMarkets/tokenManager/IGMTokenManager.sol`, we see that the expectant struct encoding is `Quote(uint256 attestationId,uint256 chainId,bytes32 userId,uint8 side,address asset,uint256 price,uint256 quantity,uint256 expiration,bytes32 additionalData)`. As a result, signature verification will fail when the typehash is derived implicitly, eg. when signing using libraries like `viem` or `ethers`.

Proof of Concept: Go to the `signTypedData` example on Viem (see signing typed-data), then replace `index.tsx` with:

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom/client';
import { type Address, type Hash, createWalletClient, custom } from 'viem';
import { goerli } from 'viem/chains';
import 'viem/window';

const walletClient = createWalletClient({
  chain: goerli,
  transport: custom(window.ethereum!),
});

function Example() {
  const [account, setAccount] = useState<Address>();
  const [signature, setSignature] = useState<Hash>();

  const connect = async () => {
    const [address] = await walletClient.requestAddresses();
    setAccount(address);
  };

  const signTypedData = async () => {
    if (!account) return;
    const signature = await walletClient.signTypedData({
      account,
      domain: {
        name: 'OndoGMTokenManager',
        version: '1',
        chainId: 1,
        verifyingContract: '0xB565E44FA22A2497E38a70ee453B1be73a3d8Fc9',
      },
      types: {
        Quote: [
          { name: 'chainId', type: 'uint256' },
          { name: 'attestationId', type: 'uint256' },
          { name: 'userId', type: 'bytes32' },
          { name: 'asset', type: 'address' },
          { name: 'price', type: 'uint256' },
          { name: 'quantity', type: 'uint256' },
          { name: 'expiration', type: 'uint256' },
          { name: 'side', type: 'uint8' },
          { name: 'additionalData', type: 'bytes32' },
        ],
      },
      primaryType: 'Quote',
      message: {
        chainId: BigInt(1),
        attestationId: BigInt(1),
        userId:
          '0x48656c6c6f20576f726c64210000000000000000000000000000000000000000',
        side: 0,
        asset: '0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826',
        price: BigInt(1000000000),
        quantity: BigInt(1000000000),
        expiration: BigInt(1717000000),
        additionalData:
          '0x48656c6c6f20576f726c64210000000000000000000000000000000000000000',
      },
    });
    setSignature(signature);
  };

  if (account)
    return (
      <>
        <div>Connected: {account}</div>
        <button onClick={signTypedData}>Sign Typed Data</button>
        {signature && <div>Receipt: {signature}</div>}
      </>
    );
  return <button onClick={connect}>Connect Wallet</button>;
}

ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(
  <Example />
```

```
);
```

Then we verify the signature created with Foundry. 2 test files:

1. `SimpleVerifier.sol`, inserted in `globalMarkets/mock` folder.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.16;

import "contracts/external/openzeppelin/contracts/utils/cryptography/ECDSA.sol";
import "contracts/globalMarkets/tokenManager/IGMTokenManager.sol";
import "contracts/external/openzeppelin/contracts/utils/cryptography/EIP712.sol";

contract SimpleVerifier is EIP712 {
  bytes32 private constant QUOTE_TYPEHASH = keccak256(
      "Quote(uint256 chainId,uint256 attestationId,bytes32 userId,address asset,uint256 price,uint256
      ↪  quantity,uint256 expiration,uint8 side,bytes32 additionalData)"
    );

  constructor() EIP712("OndoGMTokenManager", "1") {}

  function verify(bytes calldata signature) public view returns (address signer) {
    IGMTokenManager.Quote memory quote = IGMTokenManager.Quote({
      chainId: 1,
      attestationId: 1,
      userId: 0x48656c6c6f20576f726c6421000000000000000000000000000000000000000,
      asset: address(0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826),
      price: 1000000000,
      quantity: 1000000000,
      expiration: 1717000000,
      side: IGMTokenManager.QuoteSide.BUY,
      additionalData: 0x48656c6c6f20576f726c6421000000000000000000000000000000000000000
    });

    bytes32 digest = _hashTypedDataV4(
      keccak256(
        abi.encode(
          QUOTE_TYPEHASH,
          quote.chainId,
          quote.attestationId,
          quote.userId,
          quote.asset,
          quote.price,
          quote.quantity,
          quote.expiration,
          quote.side,
          quote.additionalData
        )
      )
    );

    signer = ECDSA.recover(digest, signature);
  }
}
```

2. Test verification `SimpleVerifier.t.sol`:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.16;

import "contracts/globalMarkets/mock/SimpleVerifier.sol";
import "forge-std/Test.sol";


contract VerifierTest is Test {
    SimpleVerifier verifier;

    function setUp() public {
        vm.chainId(1);
        verifier = new SimpleVerifier{salt: "test"}();
    }

    function test_verifySig() public {
        assertEq(address(verifier), 0xB565E44FA22A2497E38a70ee453B1be73a3d8Fc9, "incorrect verifier
        ↪  address derivation"); // note that the address will change due to code changes, which
        ↪  affects the metadata appended
        bytes memory signature = hex"07a310b16b4071b1fbadc5fbda528f4eb3df7d00ce3a3ff015be0c35fdaa81c12e
        ↪  cc6ceb814ebaf29bf60c0d447a9bb5323043eac62151f859142286e3cc7c901b";
        address signer = 0x91105527DCA5afBFc9A6c11260CE048C24c2078d;
        address recoveredSigner = verifier.verify(signature);
        assertEq(recoveredSigner, signer);
    }
}
```

Recommendation:

- Modify `QUOTE_TYPEHASH`:

```
- Quote(uint256 chainId,uint256 attestationId,bytes32 userId,address asset,uint256 price,uint256
↪  quantity,uint256 expiration,QuoteSide side,bytes32 additionalData)
+ `Quote(uint256 chainId,uint256 attestationId,bytes32 userId,address asset,uint256 price,uint256
↪  quantity,uint256 expiration,uint8 side,bytes32 additionalData)
```

- Modify the `Quote` struct param order to match `QUOTE_TYPEHASH`:

```
struct Quote {
  uint256 chainId;
  uint256 attestationId;
  bytes32 userId;
  address asset;
  uint256 price;
  uint256 quantity;
  uint256 expiration;
  QuoteSide side;
  bytes32 additionalData;
}
```

Ondo Finance: Fixed in PR 433.

Spearbit: Fix verified.

5.2 Low Risk

5.2.1 Inconsistent `gmIdentifier` definition

Severity: Low Risk

Context: OndoComplianceGMView.sol#L33-L34, GMTokenManager.sol#L60-L62

Description: `gmIdentifier` is defined as `address(0x5ec);` in `OndoComplianceGMView`, but `address(uint160(uint256(keccak256(abi.encodePacked("global_markets")))));` = `0x428ECB70E90d1527A5f5e177789f51747b883F34` in `GMTokenManager`, which would cause confusion.

Recommendation: Use the latter definition:

```
- address public gmIdentifier = address(0x5ec);
+ address public gmIdentifier =
↪  address(uint160(uint256(keccak256(abi.encodePacked("global_markets")))));
```

Ondo Finance: Fixed in commit 33aecfb8.

Spearbit: Fix verified.

5.2.2 Missing `onUSD` refund on mints

Severity: Low Risk

Context: GMTokenManager.sol#L196-L206

Description: The `mintWithAttestation` allows paying with tokens other than `onUSD`. The issue is in that case, the user is charged the entire specified amount `depositTokenAmount` instead of the actual cost `mintOnusdValue`.

When minting with other tokens, the user's funds are first passed into the `onUSDManager` contract, which mints temporary `onUSD` tokens for the purchase. Suppose the user is using some token which can have a volatile exchange rate with respect to `onUSD` tokens. In that case, they will need to specify a higher then necessary input amount, in order to have their transaction go through reliably.

These funds are then used to mint `onUSD` tokens, which are then burnt to mint the actual GM tokens. The price for the GM tokens is calculated in the `mintOnusdValue` variable. The contract only burns `mintOnusdValue` value of `onUSD` tokens, but does not pay out the remainder of the tokens. So if users specify `depositTokenAmount` to be higher than the required amount to keep in mind the possible volatility of the payment token, they are charged the full amount even though they should only be charged `mintOnusdValue` amount.

Proof of Concept:

```
diff --git a/forge-tests/globalMarkets/tokenManager/GMTokenManagerPSMIntegrationTest.t.sol
↪  b/forge-tests/globalMarkets/tokenManager/GMTokenManagerPSMIntegrationTest.t.sol
index d480e39..2988795 100644
--- a/forge-tests/globalMarkets/tokenManager/GMTokenManagerPSMIntegrationTest.t.sol
+++ b/forge-tests/globalMarkets/tokenManager/GMTokenManagerPSMIntegrationTest.t.sol
@@ -251,9 +251,9 @@ contract onUSDManagerTest_ETH is OUSG_InstantManager_BasicDeployment {
     bytes memory signature = _createAttestation(attesterPrivateKey, quote);

     // Approve manager to spend user's onUSD
-    deal(address(USDC), user, usdcDepositAmount);
+    deal(address(USDC), user, 2 * usdcDepositAmount);
     vm.startPrank(user);
-    USDC.approve(address(gmTokenManager), usdcDepositAmount);
+    USDC.approve(address(gmTokenManager), 2 * usdcDepositAmount);

     // Token supply before subscription/burn
     uint256 onUsdSupply = onusd.totalSupply();
@@ -262,7 +262,7 @@ contract onUSDManagerTest_ETH is OUSG_InstantManager_BasicDeployment {
       quote,
       signature,
       address(USDC),
-      usdcDepositAmount
+      2 * usdcDepositAmount
     );
     vm.stopPrank();
```

Output tokens remain the same even though the input USDC amount doubles.

Recommendation: As long as `onUSD` and the `depositToken` are maintaining their peg, only `mintOnusdValue` amount of tokens should be charged (for a 1:1 peg). For unpegged `depositToken`, consider refunding the unused amount of `onUSD` tokens.

```
uint256 depositedOnusdValue = onUSDManager.subscribe(
  depositToken,
  depositTokenAmount,
  mintOnusdValue
);
uint256 refund = depositedOnusdValue - mintOnusdValue;
```

Ondo Finance: Fixed in PR 435.

Spearbit: Fix verified.

5.3 Informational

5.3.1 Typos, Minor Recommendations, Redundancies

Severity: Informational

Context: OndoComplianceGMClientUpgradeable.sol#L98, IssuanceHours.sol#L33, IssuanceHours.sol#L60, IssuanceHours.sol#L72-L83, IssuanceHours.sol#L90-L94, onUSDFactory.sol#L116-L138, IonUSDManagerEvents.sol#L17, onUSD.sol#L59, OndoSanityCheckOracle.sol#L169, OndoSanityCheckOracle.sol#L180-L186, GMTokenFactory.sol#L188-L194, GMTokenFactory.sol#L231-L234, IGMTokenManagerEvents.sol#L40-L43, TokenPauseManagerClientUpgradeable.sol#L100

Description / Recommendation:

- Typos:

```
  - accepred
  + accepted


  - Easter
  + Eastern


  - UTC-5 (5 * 3600) or UTC-4 (4 * 3600)
  + UTC-5 (-5 * 3600) or UTC-4 (-4 * 3600)
```

- Minor Recommendations:
  - Consider using BokkyPooBahDateTimeLibrary's implementation for obtaining the day of the week, which was formally verified by Zellic. The main difference is that Sunday is counted as `7` instead of `0` here, so the check is simpler as well.

```
    function checkIsValidHours() external view {
        // Computes the day of the week based on the current block timestamp
        // Ref: BokkyPooBahDateTimeLibrary's `getDayOfWeek()` implementation
        uint8 dayOfTheWeek = uint8(
          ((_abs(int256(block.timestamp) + timezoneOffset)) / DAY_IN_SECONDS + 3) %
            7 + 1
        );

        // 6 = Saturday, 7 = Sunday
        if (dayOfTheWeek > 5) {
          revert OutsideMarketHours();
        }
      }
```

  - Consider capping the offset to -12 hours / +14 hours from UTC:

    ```
    + error MaximumOffsetExceeded();
      function setTimezoneOffset(int256 _timezoneOffset) public onlyOwner {
    +   if(_timezoneOffset < -12 * HOUR_IN_SECONDS || _timezoneOffset > 14 * HOUR_IN_SECONDS)
    ↪   revert MaximumOffsetExceeded();
        emit SetTimezoneOffset(timezoneOffset, _timezoneOffset);

        timezoneOffset = _timezoneOffset;
      }
    ```

  - Modify the slot gaps for `OndoComplianceGMClientUpgradeable` and `TokenPauseManagerClientUpgradeable` to be 1 less each, so the offset stays consistent:

    ```
    - uint256[50] private __gap;
    + uint256[49] private __gap;
    ```

  - Sanity check that `bps` / `minimumLiveness` isn't 0 in `setDefaultAllowedDeviationBps()` / `setMinimumLiveness()` respectively, as:

    1. Practically it doesn't make sense to have zero price deviation / liveness.
    2. Would always be emitting the `AllowedDeviationSet()` and `AllowedDeviationSet` events when posting prices.

  - Modify `setTokenPauseManager()` to be consistent with other setters:

    ```
    if (_tokenPauseManager == address(0)) revert TokenPauseManagerCantBeZero();
    emit NewTokenPauseManagerSet(tokenPauseManager, _tokenPauseManager);
    tokenPauseManager = _tokenPauseManager;
    ```

  - `GMTokenFactory.sol` misses to implement `IGMTokenPad.sol` interface.
  - Consider using the beacon proxy pattern for `GMToken` deployments, since there are expected to be hundreds or thousands of instances deployed. More information can be found in this Rareskills article and in the OpenZeppelin foundry upgrades plugin documentation.

- Redundancies:
  - Granting the `DEFAULT_ADMIN_ROLE` to `msg.sender` is redundant in `__onUSD_init_unchained` as `__ERC20PresetMinterPauser_init_unchained` will already do it.

    ```
    - grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    ```

  - `multiexcall()` in `onUSDFactory` is redundant because it only has 1 callable function that can be executed just once.
  - `IonUSDManagerEvents` and `IonUSDManagerErrors` are defined but unused.
  - The `IGMTokenManagerEvents.OndoComplianceSet` event is defined but unused.

Ondo Finance:

- Fix typos, limit offset and utilize BokkyPooBahDateTimeLibrary.
- Sanity check oracle, disable 0 bps deviation and liveliness.
- Remove default admin role grant from onusd init unchained.
- Remove unused interfaces and events.
- Remove multiexcall from factory.
- Updated token pause logic, errors, and events.
- Implement `IGMTokenPad` (renamed to `IGMTokenFactory`) and have `GMTokenFactory` inherit it.

- Updated slot gaps.
- Beacon proxy pattern for `GMToken`.
- `accepred` -> `accepted` typo fix.

Spearbit: Fix verified.

5.3.2 Edge Cases & Technical Precautions

Severity: Informational

Context: GMToken.sol#L79-L80, GMToken.sol#L138-L153, IssuanceHours.sol#L72, onUSDManager.sol#L59, OndoSanityCheckOracle.sol#L166-L173, TokenManagerRegistrar.sol#L152-L153

Description/Recommendation:

- onUSDManager.sol#L59:
  - `OnUSD` points to `OndoComplianceGMView` to get compliance.
  - `onUSDManager` uses `BaseRWAManager` which directly points to compliance address.
  - Thus, if compliance address is ever changed for `onUSD`, change in both `OndoComplianceGMView` and `BaseRWAManager`.
- TokenManagerRegistrar.sol#L152:
  - While changing `gmTokenManager`, minter role is not revoked from old `gmTokenManager` (due to migration support).
  - Ensure that minter role is removed manually from old `gmTokenManager`.
- OndoSanityCheckOracle.sol:
  - Default Deviation changes will be applicable only for newly introduced tokens as existing tokens would already be set with old `defaultDeviationBps` or custom `allowedDeviationBps`. In case existing tokens deviation bps also require updation from old default value then it need to be done for each token individually.
  - Same goes for `minimumLiveness` set via `setDefaultMinimumLiveness`.
- IssuanceHours.sol#L72:
  - The system incorrectly permits trading on days when the stock market is closed due to festival holidays.
  - Note: As confirmed by Ondo team, after-hours trading is allowed overnight on trading day.
- GMToken.sol#L138-L153:
  - Ideally unpausing should only be allowed by `UNPAUSE_TOKEN_ROLE` but `CONFIGURER_ROLE` can simply change Pause Manager using `setTokenPauseManager` and can unpause contracts.
  - If not expected then do not allow changing Pause Manager if contract is in paused state.
- GMToken.sol:

  The current implementation does not pose any immediate issues, but we should keep the following in mind:

  1. Future Changes to `__TokenPauseManagerClientInitializable_init`:
  - If this function is updated in the future to include initialization of additional base contracts, we might need to switch to calling the full `__TokenPauseManagerClientInitializable_init` instead of the _unchained version to ensure proper initialization.
  2. Addition of New Base Contracts:
  - If a new base class is added that also calls `__OndoComplianceGMClientInitializable_init`, it could result in double initialization. In such cases, we would need to carefully manage initializer calls, possibly by relying on the _unchained variant to avoid reinitializing the same base contract more than once.

Ondo Finance: Acknowledged.

Spearbit: Acknowledged.

5.3.3 Unrestricted minting without Rate limit allowed for `ADMIN_MINT_ROLE`

Severity: Informational

Context: GMTokenManager.sol#L304-L317

Description: It was observed that:

- `ADMIN_MINT_ROLE` can mint as much GMTokens as they want without any upper cap and rate limit using `adminProcessMint` function.
- Also `adminProcessMint` function does not have pause restrictions which means `ADMIN_MINT_ROLE` can mint even when contract is paused for minting.

Recommendation:

1. Add rate limits so that `ADMIN_MINT_ROLE` can only mint till allowed cap.
2. Add `whenMintingNotPaused` if minting is not expected even with `ADMIN_MINT_ROLE` once minting is paused.

Ondo Finance: Fixed in PR 434.

Spearbit: Fix verified.