Ondo Global Markets Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Immeas

Al-Qa'qa'

July 14, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Ondo Global Markets is a new product by Ondo that enables users to tokenize publicly traded securities on-chain. KYC'd users can interact with Ondo to purchase GMTokens, which are tokenized representations of real-world securities. These tokens can then be freely traded on-chain. However, GMTokens can only be minted or burned during normal trading hours (24/5).

The system also introduces a stablecoin, `USDon`, which represents cash held by Ondo at a brokerage. This is the token used to purchase securities. A dedicated contract, `usdonManager`, facilitates swaps into `USDon` from supported stablecoins (e.g., `USDC`).

To enhance safety, the protocol includes a sanity check oracle, which ensures that prices for securities do not deviate significantly from a previously posted reference price. This prevents malicious or erroneous off-chain input from resulting in incorrect or unbacked token mints.

4.1 Actors and Roles

1. Actors

- **Off-chain service**: Executes the actual buying and selling of real-world securities. Signs attestations for minting and redeeming.
- **Ondo team**: Manages contracts, updates configurations, and oversees system integrity.
- **Users**: Must complete KYC to participate. Once approved, they can buy and sell tokenized securities.

2. Roles

- `DEFAULT_ADMIN_ROLE`: Grants/revokes all other roles and manages system-level configuration.
- `CONFIGURER_ROLE`: Configures protocol parameters and contract references.
- `DEPLOYER_ROLE`: Deploys new GMTokens.

- `ATTESTATION_SIGNER_ROLE`: Signs EIP-712-compliant attestations for mint/redeem operations. Must be held by an EOA.
- `MINTER_ROLE`: Allows minting of GMTokens and USDon. Held by `GMTokenManager` and `usdonManager`.
- `BURNER_ROLE`: Allows burning of GMTokens and USDon. Held by `GMTokenManager` and `usdonManager`.
- `PAUSER_ROLE` / `PAUSE_TOKEN_ROLE`: Authorized to pause contracts and token transfers.
- `UNPAUSER_ROLE` / `UNPAUSE_TOKEN_ROLE`: Authorized to unpause contracts and token transfers.
- `TOKEN_FACTORY_ROLE`: Used by `GMTokenFactory` to deploy new token contracts.

4.2 Key Components

- `USDon`: A stablecoin backed by cash held at a brokerage. It can only be held or transferred by compliant (non-OFAC) users. It is pausable.
- `usdonManager`: An `RWAManager` contract that enables users to swap supported stablecoins into USDon.
- `GMTokens`: Tokenized representations of public securities. Can only be held or transferred by compliant users. Pausable for safety via a shared mechanism in `TokenPauseManager`, which can pause or unpause all tokens globally or individually.
- `GMTokenManager`: Manages the buying and selling of GMTokens. Integrates with `usdonManager` to allow swaps from other stablecoins into USDon. Enforces compliance, trading hour restrictions (24/5), and sanity-checked pricing to prevent deviation from posted oracle prices.

4.3 Centralization Risks

These contracts are heavily managed by the Ondo team, so using this protocol requires a high level of trust in Ondo's operational security. If Ondo's privileged wallets were ever compromised, the consequences could be catastrophic for the protocol. Extreme care must be taken to safeguard admin keys and related infrastructure.

5 Audit Scope

All files under `contracts/globalMarkets`:

```
contracts/globalMarkets/GMTokenCompliance/OndoComplianceGMClientUpgradeable.sol
contracts/globalMarkets/GMTokenCompliance/OndoComplianceGMView.sol
contracts/globalMarkets/issuanceHours/IssuanceHours.sol
contracts/globalMarkets/onUSDManager/onUSDManager.sol
contracts/globalMarkets/sanityCheckOracle/OndoSanityCheckOracle.sol
contracts/globalMarkets/tokenFactory/GMTokenFactory.sol
contracts/globalMarkets/tokenManager/GMTokenManager.sol
contracts/globalMarkets/tokenPauseManager/TokenPauseManager.sol
contracts/globalMarkets/tokenPauseManager/TokenPauseManagerClientUpgradeable.sol
contracts/globalMarkets/BridgeRegistrarStub.sol
contracts/globalMarkets/GMToken.sol
contracts/globalMarkets/onUSD.sol
contracts/globalMarkets/onUSDFactory.sol
contracts/globalMarkets/TokenManagerRegistrar.sol
```

6 Executive Summary

Over the course of 10 days, the Cyfrin team conducted an audit on the Ondo Global Markets smart contracts provided by Ondo. In this period, a total of 12 issues were found.

The audit uncovered two low-severity issues: The first involved a missing role grant during the deployment of `USDon`. The second concerned a discrepancy between the compliance check used when minting/redeeming `USDon` and the one used during token transfers.

Several informational findings were also identified, covering best practices, code quality, and opportunities for improvement in testing and documentation.

During the audit, the team renamed the USD stablecoin from `onUSD` to `USDon`. This change was introduced in commit `85d5b09` and was deemed safe.

The Cyfrin team also contributed test enhancements, included in commit `d3155d0`.

Summary

Project Name	Ondo Global Markets
Repository	rwa-internal
Commit	a74d03f4a71b...
Audit Timeline	Jul 1st - Jul 14th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	2
Informational	10
Gas Optimizations	0
Total Issues	12

Summary of Findings

[L-1] `guardian` missing `PAUSER_ROLE` grant in `onUSD` deployment	Resolved
[L-2] Compliance check discrepancy between `onUSDManager` and `onUSD` transfers	Acknowledged
[I-1] `OndoSanityCheckOracle::setAllowedDeviationBps` is not checking zero value as input which will introduce problems using it	Resolved
[I-2] Inconsistent unpause role in `onUSD`	Resolved
[I-3] `GMTokenManager::mintWithAttestation` breaks Check-Effects-Interactions pattern	Resolved
[I-4] Inconsistent role for `GMTokenManager::setIssuanceHours`	Resolved
[I-5] Unnecessary boolean comparisons in `GMTokenManager`	Resolved

[I-6] Inconsistent type usage for `IssuanceHours.HOUR_IN_SECONDS`	Resolved
[I-7] Confusing field name `minimumLiveness` in `PriceData` struct	Resolved
[I-8] Test enhancements	Resolved
[I-9] Natspec enhancements	Resolved
[I-10] Missing `nonReentrant` modifier on `GMTokenManager` mint/redeem	Resolved

7 Findings

7.1 Low Risk

7.1.1 `guardian` missing `PAUSER_ROLE` grant in `onUSD` deployment

Description: Deployment of the `onUSD` token is handled via the `onUSDFactory`, which sets up the token as an upgradeable proxy using the transparent proxy pattern (EIP-1967).

As documented in the contract comments, the `guardian` address is expected to be granted both the `DEFAULT_ADMIN_ROLE` and `PAUSER_ROLE`:

globalMarkets/onUSDFactory.sol#L33-36

```
/**
...
 *         Following the above mentioned deployment, the address of the onUSD_Factory contract will:
 *         i) Grant the `DEFAULT_ADMIN_ROLE` & PAUSER_ROLE to the `guardian` address <<----------------
 *         ii) Revoke the `MINTER_ROLE`, `PAUSER_ROLE` & `DEFAULT_ADMIN_ROLE` from address(this).
 *         iii) Transfer ownership of the ProxyAdmin to that of the `guardian` address.
 */
```

However, in the actual deployment logic, only the `DEFAULT_ADMIN_ROLE` is granted to the `guardian`. The `PAUSER_ROLE` is omitted:

globalMarkets/onUSDFactory.sol#L88

```
  function deployonUSD( ... ) external onlyGuardian returns (address, address, address) {
    ...

    // @audit `PAUSER_ROLE` not granted to guardian
>>  onusdProxied.grantRole(DEFAULT_ADMIN_ROLE, guardian);

    onusdProxied.revokeRole(MINTER_ROLE, address(this));
    onusdProxied.revokeRole(PAUSER_ROLE, address(this));
    onusdProxied.revokeRole(DEFAULT_ADMIN_ROLE, address(this));

    onusdProxyAdmin.transferOwnership(guardian);
    assert(onusdProxyAdmin.owner() == guardian);
    initialized = true;
    emit onUSDDeployed( ... );

    return ( ... );
  }
```

As a result, deployment completes without the `guardian` address having the `PAUSER_ROLE` in the `onUSD` token contract, contrary to the intended and documented behavior.

Impact: The `guardian` will not have the `PAUSER_ROLE` in the deployed `onUSD` token contract. This prevents them from pausing the token immediately after deployment, potentially limiting their ability to respond to emergencies or enforce compliance controls. However, since the guardian retains the `DEFAULT_ADMIN_ROLE`, they can manually grant themselves the `PAUSER_ROLE` later. Still, this deviates from the intended one-step initialization flow and introduces the risk of operational oversight.

Recommended Mitigation: Grant the `PAUSER_ROLE` to the `guardian` address immediately after assigning the `DEFAULT_ADMIN_ROLE`, to match both the contract's intended behavior and its documentation:

```
  onusdProxied.initialize(name, ticker, complianceView);

  onusdProxied.grantRole(DEFAULT_ADMIN_ROLE, guardian);
+ onusdProxied.grantRole(PAUSER_ROLE, guardian);

  onusdProxied.revokeRole(MINTER_ROLE, address(this));
```

```
onusdProxied.revokeRole(PAUSER_ROLE, address(this));
```

Ondo: Fixed in commit b13a651. It's the comment that is incorrect here - we only want to grant the default admin role, as it is temporarily used by the deployment EOA to configure the contract properly. Once configured, the default admin is renounced. If the pauser was also granted to the EOA on deployment it would just require another call to renounce

Cyfrin: Verified. Comment removed.

7.1.2 Compliance check discrepancy between `onUSDManager` and `onUSD` transfers

Description: When minting or redeeming `onUSD` via `onUSDManager`, the contract extends `BaseRWAManager`, which performs a compliance check using the `onUSD` token address (`address(onUSD)`) as the `rwaToken` identifier. This happens in `BaseRWAManager::_processSubscription`:

```
// Reverts if user address is not compliant
ondoCompliance.checkIsCompliant(rwaToken, _msgSender());
```

The same check occurs during redemptions via `BaseRWAManager::_processRedemption`.

Separately, the `onUSD` token contract itself performs compliance checks inside `onUSD::_beforeTokenTransfer`, which is invoked during transfers, minting, and burning. This function calls the inherited `OndoComplianceGMClientUpgradeable::_checkIsCompliant`, which delegates to `OndoComplianceGMView::checkIsCompliant`:

```
function checkIsCompliant(address user) external override {
  compliance.checkIsCompliant(gmIdentifier, user);
}
```

Here, `OndoComplianceGMViewgmIdentifier` is a hardcoded address derived from the string `"global_markets"` and used as the `rwaToken` identifier:

```
address public gmIdentifier =
  address(uint160(uint256(keccak256(abi.encodePacked("global_markets")))));
```

As a result, minting and redeeming will trigger two compliance checks with different identifiers:

- `address(onUSD)` via the manager logic
- `gmIdentifier` via the token's `_beforeTokenTransfer`

Impact: Although `_beforeTokenTransfer` runs during minting and burning, meaning both compliance checks still occur, the use of two different `rwaToken` identifiers introduces an unnecessary inconsistency. If the two compliance lists are not aligned, minting or redeeming could revert unexpectedly, despite the user being compliant under one identifier.

Recommended Mitigation: There are two possible mitigation approaches, depending on which compliance identifier is intended as canonical for `onUSD`.

1) Update `OnUSD::_beforeTokenTransfer` to explicitly use `address(this)` as the `rwaToken` in all compliance checks. This aligns the transfer/mint/burn logic with the identifier used in the manager's mint/redeem flow, ensuring consistency and eliminating the need to maintain two separate compliance lists.

```
if (from != msg.sender && to != msg.sender) {
  compliance.checkIsCompliant(address(this), msg.sender);
}

if (from != address(0)) {
  // If not minting
  compliance.checkIsCompliant(address(this), from);
}

if (to != address(0)) {
  // If not burning
```

```
    compliance.checkIsCompliant(address(this), to);
  }
```

2) If `gmIdentifier` is intended to serve as a shared compliance identity for global markets assets (including `onUSD`), consider using `gmIdentifier` in the `onUSDManager` mint/redeem flow as well. This would unify all compliance checks under a single identifier, reducing operational fragmentation.

Ondo: Acknowledged. The `OndoCompliance` check in the `USDonManager` only exists due to the `USDonManager` inheriting the `BaseRWAManager` - since the check already exists in `USDon` transfers themselves it would be completely redundant if used. Knowing this, we will leave the sanctions and blocklist unset for `USDon` in `OndoCompliance` so that the checks coming from the `USDonManager` are effectively bypassed, and we instead rely on checks stemming from `USDon` transfers themselves and keyed on the `gmIdentifier`.

7.2 Informational

7.2.1 `OndoSanityCheckOracle::setAllowedDeviationBps` is not checking zero value as input which will introduce problems using it

Description: In `OndoSanityCheckOracle`, there are two types of deviation values: a default deviation applied to all tokens by default, and a token-specific deviation set per asset via `setAllowedDeviationBps()`.

The default deviation value is validated to be non-zero, while token-specific deviations can be set to zero:

OndoSanityCheckOracle.sol#L222-L245

```
function setAllowedDeviationBps(...) external onlyRole(CONFIGURER_ROLE) {
  if (bps >= BPS_DENOMINATOR) revert InvalidDeviationBps();
  prices[token].allowedDeviationBps = bps;
  emit AllowedDeviationSet(token, bps);
}

function setDefaultAllowedDeviationBps(...) public onlyRole(CONFIGURER_ROLE) {
  if (bps == 0) revert InvalidDeviationBps(); // enforced here
  if (bps >= BPS_DENOMINATOR) revert InvalidDeviationBps();
  emit DefaultAllowedDeviationSet(defaultDeviationBps, bps);
  defaultDeviationBps = bps;
}
```

Setting a token deviation to zero is functionally meaningless, however, because zero is interpreted as "use the default" during price posting:

OndoSanityCheckOracle.sol#L189-L192

```
if (priceData.allowedDeviationBps == 0) {
  priceData.allowedDeviationBps = defaultDeviationBps;
  emit AllowedDeviationSet(token, priceData.allowedDeviationBps);
}
```

This creates a subtle inconsistency: the contract accepts `0` as a valid input for per-token deviations, but the value will be ignored and overridden when posting a price. If zero deviation is considered too strict or unsupported, enforce a `bps > 0` check in `setAllowedDeviationBps()`, mirroring the validation in `setDefaultAllowedDeviationBps()`.

Alternatively, if `0` is meant to indicate "use default," consider introducing an explicit boolean field to track whether a token's deviation has been explicitly set, rather than relying on `0` as a sentinel value.

Ondo: Fixed in commit 6a33346

Cyfrin: Verified. `allowedDeviationBps` is not allowed to be 0.

7.2.2 Inconsistent unpause role in `onUSD`

Description: `onUSD::unpause` is restricted to `DEFAULT_ADMIN_ROLE`, unlike other contracts in the system that use a dedicated `UNPAUSER_ROLE`. This breaks consistency in access control design and limits flexibility in delegating unpause authority:

```
function unpause() public override onlyRole(DEFAULT_ADMIN_ROLE) {
  _unpause();
}
```

Consider using `UNPAUSER_ROLE` for `onUSD::unpause` to align with the pattern used across other contracts.

Ondo: Fixed in commit 650c527

Cyfrin: Verified.`UNPAUSER_ROLE` used in `USDon::unpause` (renamed)

7.2.3 `GMTokenManager::mintWithAttestation` breaks Check-Effects-Interactions pattern

Description: In `GMTokenManager::mintWithAttestation`, the function transfers tokens from the user before performing internal accounting operations such as rate limiting, burning, and minting. This violates the check-effects-interactions pattern, where external calls (like token transfers) should typically come after all internal state updates to reduce risk.

While the token being transferred is assumed to be a trusted stablecoin, this ordering increases the surface area for unexpected behavior if any integrated token misbehaves (e.g., via callback hooks, pausable logic, or fee-on-transfer behavior).

Consider reordering operations in `mintWithAttestation` to follow the check-effects-interactions pattern—performing rate limiting, burns, and mints **before** calling `token.transferFrom()`.

Ondo: Fixed in commit `29bdeb9`

Cyfrin: Verified. rate limiting now done before external calls.

7.2.4 Inconsistent role for `GMTokenManager::setIssuanceHours`

Description: The `GMTokenManager::setIssuanceHours` function is restricted to `CONFIGURER_ROLE`, whereas other configuration and role assignment functions across the system are typically restricted to `DEFAULT_ADMIN_ROLE`. This inconsistency may cause confusion about which roles are responsible for governance and configuration actions.

Consider aligning access control by restricting `setIssuanceHours` to `DEFAULT_ADMIN_ROLE`, consistent with similar configuration functions elsewhere.

Ondo: Fixed in commit `3d18299`

Cyfrin: Verified. `DEFAULT_ADMIN_ROLE` is now used for `GMTokenManager::setIssuanceHours`.

7.2.5 Unnecessary boolean comparisons in `GMTokenManager`

Description: Both in `GMTokenManager::_verifyQuote#L329` and `GMTokenManager::adminProcessMint#L389` there's a boolean comparison:

```
if (gmTokenAccepted[gmToken] == false) revert GMTokenNotRegistered();
```

This is redundant. Consider simplifying it to:

```
if (!gmTokenAccepted[gmToken]) revert GMTokenNotRegistered();
```

Ondo: Fixed in commit `1877211`

Cyfrin: Verified.

7.2.6 Inconsistent type usage for `IssuanceHours.HOUR_IN_SECONDS`

Description: In `IssuanceHours` the constant `IssuanceHours.HOUR_IN_SECONDS` field is declared as `uint`, while the rest of the codebase consistently uses `uint256`:

```
/// Constant for the number of seconds in an hour
uint constant HOUR_IN_SECONDS = 3_600;
```

Consider updating the field to use `uint256` to align with the project's standard type declarations.

Ondo: Fixed in commit `fe452a1`

Cyfrin: Verified. `HOUR_IN_SECONDS` uses type `int256` (since that removes a cast in `_validateTimezoneOffset`)

7.2.7 Confusing field name `minimumLiveness` in `PriceData` struct

Description: The `PriceData` struct in `OndoSanityCheckOracle` includes a field named `minimumLiveness`, which actually represents the maximum age a price can be before it's considered stale. The current name may be misleading, as "minimum liveness" implies a lower bound on freshness rather than an upper bound on staleness.

Consider renaming the field to something clearer like `maxPriceAge` or `staleThreshold` to better reflect its purpose and improve code readability.

Ondo: Fixed in commits `b453b57` and `9af9735`

Cyfrin: Verified. Renamed to `maxTimeDelay`.

7.2.8 Test enhancements

Description: * `GMIntegrationTest_GM_ETH`: Both tests `test_hitRateLimits_onUSDInGMFlow_Subscribe` and `test_hitRateLimits_onUSDInGMFlow_Redeem` have empty `expectReverts`:

```
// Should fail due to onUSD rate limit
vm.expectRevert();
gmTokenManager.mintWithAttestation(
  quote,
  signature,
  address(USDC),
  usdcAmount
);
```

Accepting any revert could hide unexpected errors allowing bugs to still pass the tests. Consider catching the expected revert:

```
    // Should fail due to onUSD rate limit
-   vm.expectRevert();
+   vm.expectRevert(OndoRateLimiter.RateLimitExceeded.selector);
    gmTokenManager.mintWithAttestation(
      quote,
      signature,
      address(USDC),
      usdcAmount
    );
```

- `GmTokenManagerSanityCheckOracleTest`: The test `testPostPricesWithInvalidInput` also has an empty `expectRevert()`. This test should ideally be split into two, `...WithInvalidToken`, `...WithInvalidPrice` and expect the correct errors: `InvalidAddress` and `PriceNotSet`.
- `error TokenPauseManagerClientUpgradeable.TokenPauseManagerCantBeZero` lacks a test. Consider adding one for assigning an invalid `TokenPauseManager`.
- `GMTokenManagerTest_ETH`: The test `testMintFromNonKYCdSender` mentions a "KYC role" which doesn't exist. It also catches an empty revert on L626. This catch does not catch the correct error, it catches a `OneRateLimiter.RateLimitExceeded` error since the user has no rate limit config. Since the user is added to the registry on L601, effectively saying it's KYC'd. Thus it passes the KYC check. Consider removing mentions of a KYC role, catching the correct revert (`IGMTokenManagerErrors.UserNotRegistered`) and remove the addition of the user to the registry.

Cyfrin: Fixed by Cyfrin in commit `d3155d0`

7.2.9 Natspec enhancements

Description: * `onUSD_Factory::deployonUSD` is missing the `complianceView` parameter in its natspec.

- `onUSD_Factory.onUSDDeployed` event is missing parameters `name`, `ticker`, and `complianceView`
- `GMTokenManager::constructor` is missing `_onUsd` parameter

- `GMTokenManager::adminProcessMint` is missing `gmToken` parameter
- `TokenPauseManager::unpauseAllTokens`: the text `Only affects tokens paused by the pauseAllTokens function` could be worded better as this is *all* tokens.

Ondo: Fixed in commit `d7dc414`

Cyfrin: Verified.

7.2.10 Missing `nonReentrant` modifier on `GMTokenManager` mint/redeem

Description: The `GMTokenManager::mintWithAttestation` and `GMTokenManager::redeemWithAttestation` functions perform external token transfers and internal state updates but do not use the `nonReentrant` modifier. While `GMTokenManager` inherits from OpenZeppelin's `ReentrancyGuard`, which is currently unused, the modifier is not applied to these functions.

Consider adding the `nonReentrant` modifier to `mintWithAttestation` and `redeemWithAttestation`.

Ondo: Fixed in commit `d7dc414`

Cyfrin: Verified.