Ondo: GM Solana

Security Review

Cantina Managed review by:
J4X, Lead Security Researcher
Mario Poneder, Security Researcher
N4nika, Security Researcher

December 18, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

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

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

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

From Nov 20th to Dec 2nd the Cantina team conducted a review of gm-solana on commit hash 3f96676f. The team identified a total of **59** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	4	4	0
Medium Risk	8	8	0
Low Risk	30	30	0
Gas Optimizations	0	0	0
Informational	16	15	1
Total	**59**	**58**	**1**

2.1 Scope

The security review had the following components in scope for gm-solana on commit hash 3f96676f:

```
programs/ondo-gm
├── Cargo.toml
├── proptest-regressions
│   └── instructions
│       └── token_manager.txt
├── src
│   ├── constants.rs
│   ├── errors.rs
│   ├── events.rs
│   ├── instructions
│   │   ├── close_attestation_account.rs
│   │   ├── gm_token_admin_operations.rs
│   │   ├── gm_token_factory_admin_operations.rs
│   │   ├── gm_token_manager_admin_operations.rs
│   │   ├── initialize_user.rs
│   │   ├── mod.rs
│   │   ├── role_operations.rs
│   │   ├── sanity_checker_admin_operations.rs
│   │   ├── token_factory.rs
│   │   ├── token_limit_admin_operations.rs
│   │   ├── token_manager.rs
│   │   ├── update_scaled_ui_multiplier.rs
│   │   ├── usdc_swap_context.rs
│   │   ├── usdon_admin_operations.rs
│   │   ├── usdon_manager_admin_operations.rs
│   │   ├── usdon_swap_context.rs
│   │   └── whitelist_operations.rs
│   ├── lib.rs
│   ├── state
│   │   ├── attestation.rs
│   │   ├── gmtoken_manager_state.rs
│   │   ├── mod.rs
│   │   ├── ondo_user.rs
│   │   ├── roles.rs
│   │   ├── sanity_check.rs
│   │   ├── token_limit.rs
│   │   ├── usdon_manager_state.rs
```

```
│   │   └── whitelist.rs
│   └── utils
│       ├── capacity.rs
│       ├── decimals.rs
│       ├── mod.rs
│       └── mul_div.rs
└── Xargo.toml
```

3 Findings

3.1 Critical Risk

3.1.1 Infinite mint/redeem possible through signature manipulation

Severity: Critical Risk

Context: token_manager.rs#L250-L300

Description: The ondo protocol uses attestations to ensure that no unrestricted minting is possible. With each call to “.

The table below is taken from the official solana_secp256k1_program documentation

Index	Bytes	Type	Description
0	2	u16	`signature_offset` — offset to 64-byte signature plus 1-byte recovery ID.
2	1	u8	`signature_offset_instruction_index` — within the transaction, the index of the transaction whose instruction data contains the signature.
3	2	u16	`eth_address_offset` — offset to 20-byte Ethereum address.
5	1	u8	`eth_address_instruction_index` — within the transaction, the index of the instruction whose instruction data contains the Ethereum address.
6	2	u16	`message_data_offset` — offset to start of message data.
8	2	u16	`message_data_size` — size of message data in bytes.
10	1	u8	`message_instruction_index` — within the transaction, the index of the instruction whose instruction data contains the message data.

In this table, we can see that each part of the struct also has a corresponding `instruction_index`. However, looking at the code below, we can see that these are ignored during parsing and never checked afterward.

```
// Parse offsets (all u16 are little-endian):
let sig_off = u16::from_le_bytes([data[rd], data[rd + 1]]) as usize;
let eth_off = u16::from_le_bytes([data[rd + 3], data[rd + 4]]) as usize;
let msg_off = u16::from_le_bytes([data[rd + 6], data[rd + 7]]) as usize;
let msg_len = u16::from_le_bytes([data[rd + 8], data[rd + 9]]) as usize;

msg!(
" Sig offset: {}, ETH offset: {}, Message offset: {}, length: {}",
sig_off,
eth_off,
msg_off,
msg_len
);
require!(msg_len == 32, SecpError::WrongDigestLen);
require!(msg_off + msg_len <= data.len(), SecpError::MalformedSecpIx);
require!(eth_off + 20 <= data.len(), SecpError::MalformedSecpIx);
```

As a result, the `secp` instruction could contain the intended digest. However, the actual verification performed by the `secp256k1_program` was based on data from a prior instruction. This prior instruction can be an instruction to a user-controlled program that contains a digest, a valid signature of the digest by the attacker, and the attacker's eth address at the same offsets as in the `secp256k1_program`. This way, both would pass, and the program would mint, while no actual signing happened at all.

Proof of Concept: An exemplary attack could look like this. The user creates a custom program that accepts any input. He structures a batch of instructions as follows:

IX-Index	program	ix	content

0	usersCustomProgram	`acceptAllInputIx`	**1:** Signature of `randomMessage32Bytes` by `attackerEthAddress`. **2:** `attackerEthAddress`. **3:** randomMessage32Bytes. The parts need to be spaced according to the offsets specified in the table above
1	secp256k1_program	`secp256k1`	This will contain a header with some variable offsets, and for each of the parts, the `instruction_index` will point to IX0. In the instructions data, the correct digest and eth pubkey of ondo must be placed at the correct offsets, the signature can be any value as it will never be checked
2	ondo-gm	`mint_with_attestation`	This will pass as `secp256k1` passed and contained the correct digest and eth address within its ix data

By using this bundled IX, the attacker can mint/redeem as often as they want without any attestations.

Recommendation: We recommend ensuring that the `instruction` indexes equal the instruction currently being checked.

Ondo Finance: Fixed in commit 12e4d3c1. We'll only support "inline" mode, the secp256k1 instruction must reference itself and must contain all calldata. There must be *at least one* instruction containing a valid signature for the message digest calculated in the program, and the recovered ETH address must match what's stored in state.

Cantina Managed: Fix verified.

3.2 High Risk

3.2.1 Privilege escalation via lamport transfer to role PDA in whitelist operations

Severity: High Risk

Context: whitelist_operations.rs#L12-L62, whitelist_operations.rs#L69-L116

Description: The `AddToWhitelist` and `RemoveFromWhitelist` instructions in `whitelist_operations.rs` use `UncheckedAccount` for role validation, allowing privilege escalation by sending SOL to a precomputed PDA address. Implementation details:

```
#[account(
    seeds = [RoleType::ADMIN_ROLE_WHITELIST, admin.key().as_ref()],
    bump
)]
pub role_account: UncheckedAccount<'info>,

// In function:
require_gt!(self.role_account.lamports(), 0, ErrorCode::ConstraintAddress);
```

Anchor's `UncheckedAccount` with `seeds` only validates the address, not ownership or data. Furthermore, the code only checks lamports, not whether it's a valid `Roles` account. Attack path and impact:

1. Attacker precomputes their role PDA address:

   ```
   findProgramAddressSync([b"AdminRoleWhitelist", attacker.key()], programId)
   ```

2. Attacker sends SOL to that address (simple transfer, no account creation needed).
3. Attacker has now gained `ADMIN_ROLE_WHITELIST` privileges.

This results in full control over the mint/redeem operations by whitelisting malicious/sanctioned addresses or removing genuine addresses from the whitelist at will, effectively bypassing compliance controls and user rate limiting, as well as blocking users from swapping.

Furthermore, the attacker can steal rent by closing `Whitelist` accounts, which is 0.12 USD at the current price, i.e. 120k USD per million whitelisted users.

Recommendation: It is recommended to change `role_account` from `UncheckedAccount` to `Account<Roles>`, e.g.:

```
/// The Roles account verifying the admin has ADMIN_ROLE_WHITELIST
/// # PDA Seeds
/// - ADMIN_ROLE_WHITELIST
/// - Admin's address
#[account(
    seeds = [RoleType::ADMIN_ROLE_WHITELIST, admin.key().as_ref()],
    bump,
)]
pub roles: Account<'info, Roles>,  // Changed from UncheckedAccount
```

Ondo Finance: Fixed in commit f4620ef0.

Cantina Managed: Fix verified.

3.2.2 Unchecked `confidence` value allows for usage of non trustworthy oracle prices

Severity: High Risk

Context: token_manager.rs#L718-L738

Description: The protocol uses the Pyth oracle to ensure that no USDC depeg above 10% has happened when minting USDon for USDC.

```
let usdc_price = match usdc_price_update_info.key() {
    USDC_PYTH_ORACLE_ADDRESS => {
        // Fetch the feed ID for the USDC token price from its hex representation.
        let usdc_feed_id: [u8; 32] = get_feed_id_from_hex(USDC_PYTH_ID)?;

        // Deserialize `usdc_price_update_info` account data into PriceUpdateV2 struct
        let data = usdc_price_update_info.try_borrow_data()?;
        let usdc_price_update_data = PriceUpdateV2::try_deserialize(&mut &data[..])?;

        // Retrieve current USDC/USD price from Pyth oracle with freshness validation
        // This ensures we're using recent price data to prevent stale price attacks
        usdc_price_update_data
            .get_price_no_older_than(
                &Clock::get()?,
                self.usdon_manager_state.oracle_price_max_age,
                &usdc_feed_id,
            )?
            .price
    }
    _ => return err!(OndoError::UsdcOracleNotImplemented),
};

// Validate that USDC price is above minimum threshold
require_gte!(usdc_price, MIN_PRICE, OndoError::UsdcBelowMinimumPrice);
```

The `get_price_no_older_than()` function will return a `Price` struct.

```
/// A Pyth price.
/// The actual price is `(price ± conf)* 10^exponent`. `publish_time` may be used to
 ↪  check the recency of the price.
#[derive(PartialEq, Debug, Clone, Copy)]
pub struct Price {
    pub price: i64,
    pub conf: u64,
```

```
    pub exponent: i32,
    pub publish_time: i64,
}
```

In this struct the `conf` value means how far the confidence interval out of all the reported prices ranges. If this value is high it indicates a non clear price. As a result this price should not be trusted if the value passes a chosen threshold.

This could lead to a non trustworthy price being used to still mint USDon while USDc might have deppeged significantly more severely.

Recommendation: We recommend checking the confidence value to be less than a value fitting the intended risk profile (5% is recommended in some tutorials).

Ondo Finance: Fixed in commit aeeb2f1e.

Cantina Managed: Fix verified.

3.2.3 Too low `MIN_PRICE` will lead to Ondo incurring significant losses in case of a USDC depeg

Severity: High Risk

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

Description: The protocol implements safeguards to protect itself against a potential USDC depeg. Before interactions using USDC, the current Pyth price of USDC is fetched and checked to ensure it is not lower than `MIN_PRICE`.

```
// Validate that USDC price is above minimum threshold
require_gte!(usdc_price, MIN_PRICE, OndoError::UsdcBelowMinimumPrice);
```

`MIN_PRICE` is currently set to 90c.

```
/// Minimum price threshold for USDC (in scaled units)
pub const MIN_PRICE: i64 = 90_000_000;
```

However, as a result, this will allow people to mint tokens for a potentially undercollateralized depegged USDC token until its price has crashed by 10%, which will take time. Especially for a high liquidity token like USDC, this price dump will take even longer.

As a result, this very low `MIN_PRICE` will allow for minting of USDon/GM for depegged USDC at a 1:1 ratio for a far longer than needed time.

Recommendation: We recommend setting `MIN_PRICE` significantly closer to 1 USD to ensure early pausing in the event of a depeg.

Ondo Finance: Fixed in commit c021a80e.

Cantina Managed: Fix verified.

3.2.4 Incorrect rounding direction in `mint_with_attestation`

Severity: High Risk

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

Description: In `mint_with_attestation`, the amount the user has to pay is calculated based on `price` and `amount`. This calculation incorrectly rounds down, causing the user to pay less than he needs to. The affected operations are the `mul_div` in the function's `true` case:

```
let amount_sent = mul_div(price, amount, PRICE_SCALING_FACTOR as u64)?;
```

as well as both `mul_div` and `normalize_decimals` in the `false` case:

```
let normalized_amount =
    normalize_decimals(amount, ctx.mint.decimals, usdc_mint_decimals)?;
```

```rust
// Calculate the amount of USDC to be sent based on the price
let amount_sent = mul_div(price, normalized_amount, PRICE_SCALING_FACTOR as u64)?;
```

The impact is very miniscule in the `true` case (but still incorrect and needs to be fixed). In the `false` case, however, especially the rounding of `normalize_decimals` can be exploited in edge cases.

The highest impact can be achieved when a user buys a very small amount of a very expensive stock. Looking at the ondo dashboard, the most expensive stock is `MELIon` at a price of approximately `2000 USD / share`. Assuming any stocks may be added to ondo in the future, however, a very expensive example would be `BRK-A` at a price of approximately `760000 USD / share`.

Breaking down the calculation of `amount_sent` yields the following:

```
normalized_amount = amount / (10**3)
amount_sent = (price * normalized_amount) / 1e9
```

Taking the following values showcases the worst case senario:

- `amount = 1999`.
- `price = 760000 * 1e9`.

This yields the following results:

- With rounding:

  ```
  normalized_amount = 1999 / 10**3 = 1.999 => 1
  amount_sent = (760000*1e9 * 1) / 1e9 = 760000 = 0.76 USD
  ```

- Without rounding:

  ```
  normalized_amount = 1999 / 10**3 = 1.999
  amount_sent = (760000*1e9 * 1.999) / 1e9 = 1519240 = 1.51924 USD
  ```

As we can see, rounding down when normalizing the amount can lead to up to a 50% loss in the worst case.

Summarized, the rounding can lead to the loss of `1 millionth` of a share. With very highly priced shares this can become quite significant. It's important to note though, that the permissioned nature of the protocol (requiring attestations to mint) greatly reduces the exploitability.

Recommendation: Consider rounding up instead of down in the mentioned cases.

Specifically, in the `true` case, the `mul_div` should round up. In the `false` case, however it would make sense to first calculate the `amount_sent` and only afterwards normalize the decimals and round up there. This would minimize rounding errors since the `mul_div` after `normalize_decimals` increases the inaccuracy.

This is because the operations are currently: `divide, multiply, divide` and division before multiplication is advised against.

Reordering the operations would yield: `multiply, divide, divide`.

Ondo Finance: Fixed in commit 3b071a32.

Cantina Managed: Fix verified.

3.3 Medium Risk

3.3.1 Attestation/Token creation process can be blocked

Severity: Medium Risk

Context: token_manager.rs#L77-L84

Description: Both the `init_mint_internal` as well as the `initialize_attestation_account` use the `create_account()` function. Below, one can see the implementation of `create_account()` in the Solana program:

```
fn create_account(
    from_account_index: IndexOfAccount,
    to_account_index: IndexOfAccount,
    to_address: &Address,
    lamports: u64,
    space: u64,
    owner: &Pubkey,
    signers: &HashSet<Pubkey>,
    invoke_context: &InvokeContext,
    transaction_context: &TransactionContext,
    instruction_context: &InstructionContext,
) -> Result<(), InstructionError> {
    // if it looks like the `to` account is already in use, bail
    {
        let mut to = instruction_context
            .try_borrow_instruction_account(transaction_context, to_account_index)?;
        if to.get_lamports() > 0 {
            ic_msg!(
                invoke_context,
                "Create Account: account {:?} already in use",
                to_address
            );
            return Err(SystemError::AccountAlreadyInUse.into());
        }
```

Since this will return an error if the account holds any lamports, an attacker can precompute the attestation/mint PDA's address and send the minimum amount of lamports needed for an empty account to it. This is about 0.001 sol or about 12c at the current price. As a result, all calls to create these will revert.

Recommendation: We recommend using `allocate`, `transfer`, and `assign` manually to prevent this.

Ondo Finance: Fixed in commit 96c2c9ba.

Cantina Managed: Fix verified.

3.3.2 `initialize_user` allows for rate limit bypass

Severity: Medium Risk

Context: initialize_user.rs#L45-L46

Description: When a user tries to mint/redeem tokens the `initialize_ondo_user` will be called and set the default limits for the user.

```
#[inline(always)]
pub fn initialize_ondo_user(&mut self, bump: u8) -> Result<()> {
    if self.ondo_user.owner != self.user.key() {
        self.ondo_user.owner = self.user.key();
        self.ondo_user.mint = self.mint.key();
        self.ondo_user.rate_limit = self.token_limit_account.default_user_rate_limit;
        self.ondo_user.limit_window = self.token_limit_account.default_user_limit_window;
        self.ondo_user.mint_capacity_used = Some(0);
        self.ondo_user.mint_last_updated = None;
        self.ondo_user.redeem_capacity_used = Some(0);
        self.ondo_user.redeem_last_updated = None;
        self.ondo_user.bump = bump;

        msg!("User initialized");
    }

    Ok(())
}
```

So if the owner was already set, no limits will be set. This leads to an issue as anyone can create a user account and set the owner using `initialize_user`. This instruction also allows the user to set his limits to the max and not have any limits at all.

Recommendation: We recommend restricting the `initialize_user` ix to admins, or instead of allowing custom rate limits use the defaults.

Ondo Finance: Fixed in commit d4244f49.

Cantina Managed: Fix verified.

3.3.3 Admin is not able to grant `PAUSER_ROLE_GMTOKEN` and `UNPAUSER_ROLE_GMTOKEN` role

Severity: Medium Risk

Context: gm_token_admin_operations.rs#L68-L71

Description: The `ADMIN_ROLE_GMTOKEN` should be able to grant the roles `MINTER_ROLE_GMTOKEN`, `PAUSER_ROLE_GMTOKEN`, and `UNPAUSER_ROLE_GMTOKEN` through the `add_gmtoken_role` function.

```
require!(
    matches!(role, RoleType::MinterRoleGmtoken),
    OndoError::InvalidRoleType
);
```

However due to the check seen above, the admin can only grant one of the three.

Recommendation: We recommend also allowing for the granting of the `PAUSER_ROLE_GMTOKEN` and `UNPAUSER_ROLE_GMTOKEN` role.

Ondo Finance: Fixed in commit 0de40882.

Cantina Managed: Fix verified.

3.3.4 Value of action will be rounded down leading to bypass of limit

Severity: Medium Risk

Context: token_manager.rs#L375

Description: To calculate the USD value of a mint/redeem, the `rate_limit_check` function calculates `(price * tokenAmount) / priceScale`.

```
fn rate_limit_check(
    &mut self,
    price: u64,
    token_amount: u64,
    current_timestamp: i64,
    is_buy: bool,
) -> Result<()> {
    let amount = mul_div(price, token_amount, GM_TOKEN_SCALING_FACTOR)?;
```

However as this uses div it will round down by up to one USD per interaction. As a result mints/redeems with a value below 1USD will not affect the limits at all and all others will be rounded down, leading to more minting being possible than actually intended.

Recommendation: We recommend rounding up in the calculation.

Ondo Finance: Fixed in commit 3b071a32.

Cantina Managed: Fix verified.

3.3.5 Attestations can't be closed in edge case

Severity: Medium Risk

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

Description: The protocol implements two mechanisms to close attestation accounts.

```
#[account(
    mut,
    address = attestation.creator
```

```
)]
pub recipient: SystemAccount<'info>,
```

However, both of these require the `attestation.creator` to be a system account. This leads to issues if a PDA without data created an attestation. In that case, the attestation can be created; however, it can't be closed, as all calls to the close functions will revert. This will cause the rent funds to get stuck.

Recommendation: We recommend setting the `recipient` to an unchecked account.

Ondo Finance: Fixed in commit 60aad84c.

Cantina Managed: Fix verified.

3.3.6 GM token pauser can also pause USDon due to shared mint authority and `Pausable` extension

Severity: Medium Risk

Context: gm_token_admin_operations.rs#L254-L293, gm_token_admin_operations.rs#L327-L366

Description: The `PauseGmToken` instruction is intended to pause GM token mints, but its constraints allow it to pause any Token-2022 mint whose authority is the shared `MINT_AUTHORITY_SEED` PDA. Therefore, a holder of `PAUSER_ROLE_GMTOKEN` can call `pause_token` with `mint = usdon_mint` and successfully pause USDon as well. This contradicts the intended role-based access control separation.

Recommendation: It is recommended to restrict `PauseGmToken` (and the corresponding `ResumeGmToken`) to exclude the USDon mint explicitly, for example by adding a constraint tying in `USDonManagerState`:

```
#[account(
    mut,
    mint::authority = mint_authority,
    mint::token_program = token_program,
    constraint = mint.key() != usdon_manager_state.usdon_mint @
    ↪  OndoError::InvalidInputMint,
)]
pub mint: InterfaceAccount<'info, Mint>;

#[account(
    seeds = [USDON_MANAGER_STATE_SEED],
    bump = usdon_manager_state.bump,
)]
pub usdon_manager_state: Account<'info, USDonManagerState>;
```

Ondo Finance: Fixed in commit 6494c563.

Cantina Managed: Fix verified.

3.3.7 USDon UI multiplier can be modified by GM token `UPDATE_MULTIPLIER_ROLE`

Severity: Medium Risk

Context: update_scaled_ui_multiplier.rs#L12-L45

Description: The `UpdateScaledUiMultiplier` instruction is intended to adjust the `ScaledUiAmount` multiplier for GM tokens, but its constraints allow the same role to change the UI multiplier for USDon as well, because USDon and GM tokens share the same `MINT_AUTHORITY_SEED` PDA as mint authority. Therefore, any holder of `UpdateMultiplierRole` can call `update_scaled_ui_multiplier` on the USDon mint.

Implications:

- The displayed value of USDon in wallets, dashboards, and internal tools that respect the `ScaledUiAmount` multiplier can be arbitrarily skewed, even though the raw on-chain balances don't change. This can:
  - Confuse users and operators about actual USDon amounts.
  - Make balances appear larger/smaller in some UIs, impacting perceived solvency or P&L.

- Complicate off-chain accounting and reconciliation if some systems use scaled amounts and others use raw base units.

- It contradicts the spec which positions `UpdateMultiplierRole` as a GM-token-only concern.

Recommendation: It is recommended to restrict `UpdateScaledUiMultiplier` so it cannot target the USDon mint, for example by adding a constraint and state account:

```
#[account(
    mut,
    mint::authority = mint_authority,
    mint::token_program = token_program,
    constraint = mint.key() != usdon_manager_state.usdon_mint @
    ↪  OndoError::InvalidInputMint,
)]
pub mint: InterfaceAccount<'info, Mint>;

#[account(
    seeds = [USDON_MANAGER_STATE_SEED],
    bump = usdon_manager_state.bump,
)]
pub usdon_manager_state: Account<'info, USDonManagerState>;
```

Ondo Finance: Fixed in commit 6494c563.

Cantina Managed: Fix verified.

3.3.8 `set_ondo_user_rate_limit` uses wrong default window

Severity: Medium Risk

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

Description: When the `set_ondo_user_rate_limit` function is called, and no limit window is set for the user, the `DEFAULT_LIMIT_WINDOW` constant is used.

```
pub fn set_ondo_user_rate_limit(&mut self, rate_limit: u64) -> Result<()> {
    // Set the rate_limit field
    self.ondo_user.rate_limit = Some(rate_limit);

    // If limit_window is not set or is zero, use default 3600 seconds (1 hour)
    if self.ondo_user.limit_window.is_none() || self.ondo_user.limit_window == Some(0) {
        self.ondo_user.limit_window = Some(DEFAULT_LIMIT_WINDOW);
    }
```

However, this is potentially the wrong value, as for each gmtoken, a `default_user_limit_window` can be defined in its `TokenLimit` account.

```
// Default user limit window for this token
pub default_user_limit_window: Option<u64>,
```

Recommendation: We recommend adapting the code as follows:

```
// If limit_window is not set or is zero, use default 3600 seconds (1 hour)
if self.ondo_user.limit_window.is_none() || self.ondo_user.limit_window == Some(0) {
    if token_limit.default_user_limit_window.is_none(){
        self.ondo_user.limit_window = Some(DEFAULT_LIMIT_WINDOW);
    } else {
        self.ondo_user.limit_window = token_limit.default_user_limit_window;
    }

}
```

Ondo Finance: Fixed in commit ebca5863.

Cantina Managed: Fix verified.

3.4 Low Risk

3.4.1 Creation of attestation PDA will not account for lamport balance

Severity: Low Risk

Context: token_manager.rs#L80

Description: When creating an attestation PDA, the program will transfer the minimum balance needed for the account.

```
// Create the instruction to create the attestation account
let ix = system_instruction::create_account(
    &self.user.key(),
    &self.attestation_id_account.key(),
    Rent::get()?.minimum_balance(space),
    space as u64,
    &crate::ID,
);
```

However this doesn't account for the account potentially already holding lamports. As a result the account might actually hold more lamports than needed.

Recommendation: We recommend only transferring enough lamports so that the minimum balance is achieved.

Ondo Finance: Fixed in commit 96c2c9ba.

Cantina Managed: Fix verified.

3.4.2 Attestation can be closed 30 seconds past creation

Severity: Low Risk

Context: close_attestation_account.rs#L52-L58, close_attestation_account.rs#L124-L129

Description: The description of `close_attestation_account` documents the following:

```
/// Close a single attestation account
///
/// The attestation account must be older than 30 seconds to be closed.
/// The rent from the closed account is returned to the recipient (original creator).
```

However, when looking at the actual implementation, one can see that it actually only enforces.

```
require_gte!(
    Clock::get()?.unix_timestamp,
    self.attestation.created_at + ATTESTATION_EXPIRATION,
    OndoError::AttestationTooNew
);
```

So while the documentation states that for the valid path the requirement is `timestamp > createdAt + 30 seconds` the actual code implements `timestamp >= createdAt + 30 seconds`.

Recommendation: We recommend enforcing > instead of >=.

Ondo Finance: Fixed in commit 1b2b3e48.

Cantina Managed: Fix verified.

3.4.3 Attestations can be reused by closing and recreating attestation PDAs before expiration

Severity: Low Risk

Context: close_attestation_account.rs#L52-L58, close_attestation_account.rs#L124-L129, token_manager.rs#L63-L117, token_manager.rs#L819-L833

Description: The implementation intends to prevent attestation replay by creating a unique `Attestation` PDA per `attestation_id` on first use, and rejecting subsequent uses of the same ID. However, the combination of the attestation-closure logic and the initialization logic allows an attestation to be closed and then reused while it is still within its expiration window.

Implementation details:

1. Mint/Redeem path uses only `expiration` and does not bind it to `ATTESTATION_EXPIRATION`.

   During mint/redeem, the program enforces only:

   ```
   // Check attestation expiration
   require!(
       current_timestamp < expiration,
       OndoError::AttestationExpired
   );
   ```

   There is no check that `expiration - current_timestamp < ATTESTATION_EXPIRATION`, i.e. that the actual validity window is smaller than `ATTESTATION_EXPIRATION`.

2. Attestation account creation treats "account empty" as unused.

   The `initialize_attestation_account` function marks an attestation as consumed by creating a PDA and writing an `Attestation` struct, but it only considers an attestation "already used" if the account is non-empty and has lamports:

   ```
   if self.attestation_id_account.lamports() == 0
       || self.attestation_id_account.data_is_empty()
   {
       // create account and write Attestation { attestation_id, creator, created_at,
       ↪  bump }
       // ...
       Ok(())
   } else {
       Err(OndoError::AttestationAlreadyUsed.into())
   }
   ```

   If the PDA has 0 lamports or empty data, it is treated as unused and can be recreated.

3. Close instructions only require 30 seconds since `created_at`, not actual attestation expiry.

   The close logic uses `ATTESTATION_EXPIRATION` (30 seconds) only to gate when an attestation account can be closed:

   ```
   require_gte!(
       Clock::get()?.unix_timestamp,
       self.attestation.created_at + ATTESTATION_EXPIRATION,
       OndoError::AttestationTooNew
   );
   ```

   The path does not check the attestation's `expiration` parameter.

Attack path and impact:

Assuming an attestation which expires ≥ 30 seconds in the future (currently not rejected by program):

1. Use an attestation once (mint or redeem).
2. Wait ≥ 30 seconds (as per `ATTESTATION_EXPIRATION`).
3. Close the corresponding `Attestation` PDA (reclaiming rent).
4. Re-use the same attestation (same `attestation_id`, signature, `price`, `amount`, `expiration`) as long as `expiration` is still in the future.

This contradicts the intended replay-protection semantics that should prevent double-spending. Off-chain logic might prevent attestations expiring ≥ 30 seconds in the future, but on-chain enforcement is still insufficient.

Time drift between on-chain and off-chain system:

Additionally, note that the `expiration` field used to gate attestation validity is likely derived from the off-chain quoting system's clock, not the Solana cluster's `Clock` sysvar. This means any clock drift or skew between the off-chain signer and the Solana cluster can extend the effective on-chain lifetime of an attestation beyond what is intended off-chain, further widening the window in which a closed attestation account can be recreated and the attestation reused. Even if the off-chain system tries to enforce a short validity window, inaccurate clocks can still result in on-chain acceptance of attestations that the off-chain system considers expired, exacerbating the replay risk without any explicit user error.

Example: If the off-chain signer's clock is 10 seconds ahead, it might issue an attestation with `expiration = now_off + 30`, which on-chain looks like `expiration = now_on + 40`. This extra 10 seconds of effective on-chain validity widens the window in which a closed attestation account can be recreated and the same attestation reused.

Recommendation: It is recommended to enforce a maximum validity window on-chain. At attestation use (mint/redeem), in addition to `current_timestamp < expiration`, also check `expiration - current_timestamp < ATTESTATION_EXPIRATION`.

Ondo Finance: Fixed in commit 1b2b3e48.

Cantina Managed: Fix verified.

3.4.4 Admin can't overwrite non-zero `limit_window`

Severity: Low Risk

Context: gm_token_manager_admin_operations.rs#L555-L558

Description: The `set_ondo_user_rate_limit()` function can be used by the admin to adjust a user's rate limits.

```
pub fn set_ondo_user_rate_limit(&mut self, rate_limit: u64) -> Result<()> {
    // Set the rate_limit field
    self.ondo_user.rate_limit = Some(rate_limit);

    // If limit_window is not set or is zero, use default 3600 seconds (1 hour)
    if self.ondo_user.limit_window.is_none() || self.ondo_user.limit_window == Some(0) {
        self.ondo_user.limit_window = Some(DEFAULT_LIMIT_WINDOW);
    }

    // Initialize rate_used fields if not already set
    if self.ondo_user.mint_capacity_used.is_none() {
        self.ondo_user.mint_capacity_used = Some(0);
    }
    if self.ondo_user.redeem_capacity_used.is_none() {
        self.ondo_user.redeem_capacity_used = Some(0);
    }
```

This will only allow the admin to set the limit window to the default if it is none or 0. However if the user set it to something very low like one second, no changes can be made.

Recommendation: We recommend allowing the admin to set a custom `limit_window`.

Ondo Finance: Fixed in commit bee06b2d.

Cantina Managed: Fix verified.

3.4.5 Restriction on multisig usage as the upgrade authority

Severity: Low Risk

Context: role_operations.rs#L14-L31

Description: The `InitRoles` context struct uses the `admin` as the payer for the rent used to create the respective role account. This admin is enforced to be the upgrade authority of the program.

Since this account holds a lot of power, it is common practice to use a multisig in its place.

```
#[account(mut)]
pub admin: Signer<'info>,

// [...]
#[account(
    init,
    payer = admin,
    space = Roles::INIT_SPACE,
    seeds = [role.seed(), user.key().as_ref()],
    bump
)]
pub roles: Account<'info, Roles>,
```

Solana restricts lamport transfers using the `system_program::transfer` instruction from deducting lamports from accounts holding data. It is important to note that anchor's `payer` constraint uses this instruction to transfer lamports from the payer to the newly created account.

This means any multisig which uses data-holding accounts as the signer is not usable with the program.

Of the currently popular solana multisigs which are open source, `Goki` uses a PDA holding data as the signer for multisig transactions.

Recommendation: Consider adding a separate `payer` account to the `InitRoles` struct, using it solely to pay for the `roles` account's rent.

```
#[account(mut)]
pub admin: Signer<'info>,

#[account(mut)]
pub payer: Signer<'info>,

// [...]
#[account(
    init,
    payer = payer,
    space = Roles::INIT_SPACE,
    seeds = [role.seed(), user.key().as_ref()],
    bump
)]
pub roles: Account<'info, Roles>,
```

Ondo Finance: Fixed in commit c62cb0a9.

Cantina Managed: Fix verified.

3.4.6 GM token minter bypasses token-level mint limits

Severity: Low Risk

Context: gm_token_admin_operations.rs#L169-L178

Description: The `GmTokenMinter::mint_gm` instruction allows any holder of `MINTER_ROLE_GMTOKEN` to mint arbitrary amounts of a GM token without enforcing the configured token-level mint limits.

In the `GmTokenMinter` account context.

```
#[account(
    seeds = [TOKEN_LIMIT_ACCOUNT_SEED, token_limit_account.mint.as_ref()],
    bump = token_limit_account.bump,
    has_one = mint @ OndoError::InvalidInputMint
)]
pub token_limit_account: Account<'info, TokenLimit>,
```

the `token_limit_account` is required and correctly bound to the `mint` account via `has_one = mint`, but it is never read or enforced in `mint_gm`.

As a result:

- Any `MINTER_ROLE_GMTOKEN` holder can mint unbounded amounts of a GM token, regardless of configuration in the `TokenLimit` account.
- The presence of `token_limit_account` in the account context gives a false sense of enforcement, but it is effectively ignored in the logic.
- This creates a privileged-path minting bypass relative to the rate/limit controls enforced in other flows.

Recommendation: It is recommended to load and enforce the `TokenLimit` configuration before calling `mint_to` in `GmTokenMinter::mint_gm` using the same capacity/rate-limit helpers as in other flows.

Ondo Finance: Fixed in commit e0e9087e.

Cantina Managed: Fix verified.

3.4.7 USDon token minter/burner bypasses token-level mint limits

Severity: Low Risk

Context: usdon_admin_operations.rs#L121-L130, usdon_admin_operations.rs#L219-L228

Description: Both `USDonMinter::mint_usdon` and `USDonBurner::burn_usdon` require a `TokenLimit` PDA in their account context, but never actually use it to enforce any limits. For example, in the `USDonMinter` account context.

```
#[account(
    seeds = [TOKEN_LIMIT_ACCOUNT_SEED, token_limit_account.mint.as_ref()],
    bump = token_limit_account.bump,
    has_one = mint @ OndoError::InvalidInputMint
)]
pub token_limit_account: Account<'info, TokenLimit>,
```

the `token_limit_account` is never read, so any holder of `MINTER_ROLE_USDON` or `ADMIN_ROLE_USDON` (likely intended in the admin case) can mint unbounded USDon regardless of the configured `TokenLimit` for that mint. The same pattern exists for `USDonBurner` with `BURNER_ROLE_USDON`.

Recommendation: It is recommended to load and enforce the `TokenLimit` state before calling `mint_to` in `USDonMinter::mint_usdon`, using the same capacity/rate-limit helpers as in other flows. Apply similar logic to `USDonBurner::burn_usdon`. Reconsider if the `ADMIN_ROLE_USDON` should still be able to bypass these limits.

Ondo Finance: Fixed in commit e0e9087e.

Cantina Managed: Fix verified.

3.4.8 Non ATA `usdc_vault` / `usdon_vault` will lead to all swaps reverting

Severity: Low Risk

Context: usdc_swap_context.rs#L139-L146

Description: Both the `initialize_usdon_manager` and `set_usdc_vault` functions allow for setting the `usdc_vault` account to an arbitrary address.

```
pub fn set_usdc_vault(&mut self, new_usdc_vault: Pubkey) -> Result<()> {
    // Validate the new USDC vault address
    require!(
        new_usdc_vault != Pubkey::default(),
        OndoError::InvalidTokenAccount
    );

    // Set the new USDC vault address
    self.usdon_manager_state.usdc_vault = new_usdc_vault;

    Ok(())
}
```

However, when the address is actually used in the `UsdcSwapContext`, it must be the ATA of `usdon_manager_state`.

```
/// The USDC vault storing USDC tokens received from users during swaps
#[account(
    mut,
    associated_token::mint = usdc_mint,
    associated_token::authority = usdon_manager_state,
    constraint = usdc_vault.key() == usdon_manager_state.usdc_vault
)]
pub usdc_vault: Box<InterfaceAccount<'info, TokenAccount>>,
```

This will cause all USDC actions to revert if the `usdc_vault` address is set to any address other than the ATA of `usdon_manager_state`. The same issue occurs for the `usdon_vault`.

Recommendation: We recommend restricting `initialize_usdon_manager`, `set_usdc_vault`, and `set_usdon_vault` to only allow for the ATA of `usdon_manager_state`.

Ondo Finance: Fixed in commit ef33da06.

Cantina Managed: Fix verified.

3.4.9 `swap_usdon_to_usdc()` allows for 0 value swap

Severity: Low Risk

Context: token_manager.rs#L656-L658

Description: The swap functions are intended not to allow for zero-value swaps. To ensure this, the following check is added at the start of both.

```
// Validate that input amount is greater than zero
 require_gt!(amount_in, 0);
```

However, in the `swap_usdon_to_usdc()`, the decimal conversion will downcast any `value < 1000` to 0, and thus still allow for a zero value swap while passing the first check.

```
// Normalize decimals from USDon (9 decimals) to USDC (6 decimals)
let normalized_amount_out =
    normalize_decimals(amount_in, self.usdon_mint.decimals, usdc_mint.decimals)?;
```

Recommendation: We recommend ensuring that `normalized_amount_out > 0` before continuing.

Ondo Finance: Fixed in commit b3160db6.

Cantina Managed: Fix verified.

3.4.10 Missing oracle-based swap pricing logic, swaps are always 1:1 despite `oracle_price_enabled`

Severity: Low Risk

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

Description: The spec describes an oracle mode for USDC↔USDon swaps where the exchange rate is derived from the Pyth USDC price and a scaling factor:

```
price_ratio = (usdc_price * PRICE_SCALING_FACTOR) / usdon_price
amount_out = (amount_in * price_ratio) / PRICE_SCALING_FACTOR
```

Additionally, there is a fixed mode with `amount_out = amount_in`. It also states that `oracle_price_enabled` toggles between "oracle vs fixed pricing".

Implementation:

- `USDonManagerState` exposes the expected configuration.
- The swap functions in `token_manager.rs` ignore the oracle price for rate calculation and always perform a nominal 1:1 swap (only decimal-normalized).

- `usdc_oracle_sanity_check` only enforces freshness and a minimum USDC price, not a rate.

Consequences:

- Swaps are effectively always at 1:1 token units (adjusted for decimals), regardless of the oracle price.
- `oracle_price_enabled` behaves as a boolean gate for a sanity check, not as a mode switch between oracle and fixed pricing.
- The documented oracle-mode formula and the distinction between "oracle vs fixed" pricing are not implemented on-chain.

Recommendation: It is recommended to implement the documented oracle-based pricing in `swap_usdc_to_usdon` and `swap_usdon_to_usdc`.

Ondo Finance: Fixed in commit 66404927.

Cantina Managed: Fix verified.

3.4.11 Pyth oracle sanity check ignores price exponent

Severity: Low Risk

Context: token_manager.rs#L718-L738

Description: In the USDC oracle sanity check, the code reads the Pyth V2 `PriceUpdateV2` and uses only the raw `price` field, ignoring the associated exponent:

- Pyth prices are represented as `(price, expo)`; the real price** is `price * 10^expo`.
- Here, only `price` is compared to `MIN_PRICE`, assuming `MIN_PRICE` is encoded using the same exponent as the current USDC feed.
- This works today only because `MIN_PRICE` was chosen to match the current feed's exponent, but it is fragile in case of future changes. Then the comparison `require_gte!(usdc_price, MIN_PRICE, ...)` can become semantically wrong, either failing valid prices or accepting under-priced USDC, undermining the intended price floor.

Recommendation: It is recommended to explicitly handle the price exponent when performing sanity checks.

Ondo Finance: Fixed in commit 80979649.

Cantina Managed: Fix verified.

3.4.12 Attestation signer address is not validated

Severity: Low Risk

Context: token_manager.rs#L129-L140

Description: The attestation verification logic reads the expected Ethereum address directly from `GmTokenManagerState.attestation_signer_secp` without checking that it has been configured to a non-zero value:

```
// Get the expected Ethereum address from the gmtoken manager state
let eth_address = self.gmtoken_manager_state.attestation_signer_secp;
```

This field defaults to `[0u8; 20]` when `GmTokenManagerState` is first created, and there is no guard that rejects the all-zero address. If the manager is initialized (or later updated) without setting a valid signer, the program will still treat the state as "configured", but no real attestation can ever pass the secp check, effectively creating a self-inflicted DoS of all trading flows that rely on attestations.

Recommendation: It is recommended to add a validation that the address is non-zero:

```
require!(
    eth_address != [0u8; 20],
    OndoError::AttestationSignerEthAddressNotSet
);
```

Ondo Finance: Fixed in commit ae36dd5b.

Cantina Managed: Fix verified.

3.4.13 Minting/Redeeming can not be used with non-ATA token accounts

Severity: Low Risk

Context: usdc_swap_context.rs#L166-L170, usdc_swap_context.rs#L166-L171, usdc_swap_context.rs#L182-L187, usdc_swap_context.rs#L182-L188, usdon_swap_context.rs#L145-L150, usdon_swap_context.rs#L145-L151

Description: Both the `UsdcSwapContext` and the `USDonSwapContext` restrict the token accounts provided by the users to be their corresponding ATAs.

```
/// The user's USDon token account
#[account(
    mut,
    associated_token::mint = usdon_mint,
    associated_token::authority = user,
    associated_token::token_program = token_program,
)]
pub user_usdon_token_account: Box<InterfaceAccount<'info, TokenAccount>>,
```

This blocks users from paying with regular token accounts.

Recommendation: We recommend allowing also for non ATA accounts.

Ondo Finance: Fixed in commit e7aafec8.

Cantina Managed: Fix verified.

3.4.14 USDC accounts don't verify correct token program

Severity: Low Risk

Context: usdc_swap_context.rs#L139-L146, usdc_swap_context.rs#L166-L171

Description: Neither the `user_usdc_token_account` nor the `usdc_vault` verifies that the SPL token program owns them.

```
/// The user's USDC token account
#[account(
    mut,
    associated_token::mint = usdc_mint,
    associated_token::authority = user,
)]
pub user_usdc_token_account: Box<InterfaceAccount<'info, TokenAccount>>,
```

```
#[account(
    mut,
    associated_token::mint = usdc_mint,
    associated_token::authority = usdon_manager_state,
    constraint = usdc_vault.key() == usdon_manager_state.usdc_vault
)]
pub usdc_vault: Box<InterfaceAccount<'info, TokenAccount>>,
```

For all other token2022 accounts, this is implemented.

Recommendation: We recommend adding a constraint that checks the token program.

Ondo Finance: Fixed in commit e7aafec8.

Cantina Managed: Fix verified.

3.4.15 `oracle_price_max_age` not checked against `MAX_AGE_UPPER_BOUND` on initialization

Severity: Low Risk

Context: usdon_manager_admin_operations.rs#L66

Description: When the `oracle_price_max_age` gets updated in the `set_oracle_price_max_age` function, it is checked as follows.

```
// Validate the new oracle price max age
require_gt!(oracle_price_max_age, 0, OndoError::InvalidOraclePriceMaxAge);

// Ensure it does not exceed the upper bound
require_gte!(
    MAX_AGE_UPPER_BOUND,
    oracle_price_max_age,
    OndoError::InvalidOraclePriceMaxAge
);
```

This ensures that `0 <oracle_price_max_age <= MAX_AGE_UPPER_BOUND`. However, the check on initialization is only this:

```
require_gt!(oracle_price_max_age, 0, OndoError::InvalidOraclePriceMaxAge);
```

This only enforces `0 <oracle_price_max_age` with no upper bound. As a result the price could actually be greater than `MAX_AGE_UPPER_BOUND`.

Recommendation: We recommend adding a check for the `oracle_price_max_age <= MAX_AGE_UPPER_BOUND`.

Ondo Finance: Fixed in commit e0002566.

Cantina Managed: Fix verified.

3.4.16 `usdc_price_update` could be zero

Severity: Low Risk

Context: usdon_manager_admin_operations.rs#L80

Description: When updating the `usdc_price_update` using the `set_usdc_price_update_address` the program ensures that the address is not zero.

```
pub fn set_usdc_price_update_address(
    &mut self,
    new_price_update_address: Pubkey,
) -> Result<()> {
    // Validate the new price update address
    require!(
        new_price_update_address != Pubkey::default(),
        OndoError::InvalidOraclePriceAddress
    );
```

However, in the initializer, any address is passed without a check against it being zero.

```
// Write data to the USDonManagerState account
self.usdon_manager_state.set_inner(USDonManagerState {
    owner: self.admin.key(),
    usdon_mint,
    oracle_price_enabled,
    oracle_price_max_age,
    usdc_price_update,
    usdc_vault,
    usdon_vault,
    bump: bumps.usdon_manager_state,
});
```

Recommendation: We recommend verifying that the `usdc_price_update != Pubkey::default()`.

Ondo Finance: Fixed in commit e0002566.

Cantina Managed: Fix verified.

3.4.17 Zero `usdon_mint` can be intialized

Severity: Low Risk

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

Description: The initializer of the `InitializeUSDonManager` verifies all addresses besides the `usdon_mint` to be non-zero.

```
// Validate vault addresses
require!(
    usdc_vault != Pubkey::default() && usdon_vault != Pubkey::default(),
    OndoError::InvalidVault
);

// Write data to the USDonManagerState account
self.usdon_manager_state.set_inner(USDonManagerState {
    owner: self.admin.key(),
    usdon_mint,
    oracle_price_enabled,
    oracle_price_max_age,
    usdc_price_update,
    usdc_vault,
    usdon_vault,
    bump: bumps.usdon_manager_state,
});
```

This could allow for accidentally intializing the account with a 9 mint, which also couldn't be changed post deployment.

Recommendation: We recommend checking that `usdon_mint != 0` in the intializer.

Ondo Finance: Fixed in commit e0002566.

Cantina Managed: Fix verified.

3.4.18 USDC mint constraint is commented out in `UsdcSwapContext`

Severity: Low Risk

Context: usdc_swap_context.rs#L158-L163

Description: In the USDC swap context, the `usdc_mint` account is intended to be constrained to the USDC mint on mainnet, but the constraint is commented out with a ”remember to uncomment for mainnet” comment:

```
/// The USDC mint (SPL Token)
#[account(
    mint::token_program = spl_token_program,
    //constraint = usdc_mint.key() == USDC_MINT uncomment for mainnet (use for
    ↪  devnet/testnet)
)]
pub usdc_mint: Box<InterfaceAccount<'info, Mint>>,
```

This relies on a manual code edit to enforce the correct USDC mint, which is error-prone and easy to forget, especially across deployments or refactors.

Recommendation: It is recommended to replace the commented constraint with a compile-time configuration using Cargo features or explicit environment flags. For example:

```
#[account(
    mint::token_program = spl_token_program,
    #[cfg(feature = "mainnet")]
    constraint = usdc_mint.key() == USDC_MINT @ OndoError::InvalidUsdcMint
```

```
)]
pub usdc_mint: Box<InterfaceAccount<'info, Mint>>;
```

Ondo Finance: Fixed in commit ff7c68dd.

Cantina Managed: Fix verified.

3.4.19 GM token admin mint cannot target PDA recipients

Severity: Low Risk

Context: gm_token_admin_operations.rs#L151-L157

Description: The `GmTokenMinter` admin mint instruction cannot mint GM tokens to PDA-owned accounts, only to system-owned accounts. The recipient is constrained as a `SystemAccount<'info>`. Anchor's `SystemAccount` enforces that `user` is owned by the system program, which PDAs (owned by the program) are not.

Consequences:

- Admins cannot mint GM tokens directly to program-owned PDAs (e.g. treasury PDAs, custodial accounts, vaults), only to EOAs.
- Any desired minting to PDA-controlled addresses must go via an intermediate EOA + transfer, which may conflict with operational or compliance requirements.

Recommendation: It is recommended to relax the recipient type to allow PDAs. For example, change `pub user: SystemAccount<'info>` to a more general `UncheckedAccount<'info>` or `AccountInfo<'info>` with explicit ownership checks if needed.

Ondo Finance: Fixed in commit 421ca348.

Cantina Managed: Fix verified.

3.4.20 Incorrect mint used in `swap_usdc_to_usdon`

Severity: Low Risk

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

Description: When `swap_usdc_to_usdon` normalizes the `amount_out` to USDC decimals, the wrong mint is used for `to_decimals`.

```
let normalized_amount_out = normalize_decimals(amount_in, usdc_mint.decimals,
↪  self.mint.decimals)?;
```

The correct one would be `self.usdon_mint.decimals` since the decimals are normalized from `USDC` to `USDon`. Since the decimals of `USDon` and GM tokens are currently the same, the calculation still returns the correct result but should still be fixed since it's technically incorrect.

Recommendation: Consider changing `self.mint.decimals` to `self.usdon_mint.decimals`.

Ondo Finance: Fixed in commit e34b7c8b.

Cantina Managed: Fix verified.

3.4.21 Missing mint capabilities for `AdminMintRoleGmtokenManager`

Severity: Low Risk

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

Description: According to the documentation, the `AdminMintRoleGmtokenManager` is supposed to be able to mint GM tokens.

Role	Seed	Capabilities
AdminMintRoleGmtokenManager	b"AdminMintRoleGmtokenManager"	Administrative mints

Looking at the `GmTokenMinter` context struct, however, only the `MINTER_ROLE_GMTOKEN` role can call `mint_gm`:

```
pub struct GmTokenMinter<'info> {
    /// The operator minting tokens, pays for destination account if needed
    #[account(mut)]
    pub operator: Signer<'info>,

    #[account(
        seeds = [RoleType::MINTER_ROLE_GMTOKEN, operator.key().as_ref()],
        bump = roles.bump,
    )]
    pub roles: Account<'info, Roles>,
    // [...]
}
```

Recommendation: Consider allowing the `AdminMintRoleGmtokenManager` to mint GM tokens.

Ondo Finance: Fixed in commit 029a3ffc.

Cantina Managed: Fix verified.

3.4.22 USDon guardian cannot remove roles after giving them

Severity: Low Risk

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

Description: Looking at how the `RoleType::MinterRoleUsdon`, `RoleType::PauserRoleUsdon` and `RoleType::BurnerRoleUsdon` are used, it is apparent that those roles can be granted by the `GUARDIAN_USDON` but not revoked by him anymore since the only instruction using these roles is `init_usdon_roles`.

```
pub fn init_usdon_roles(&mut self, role: RoleType, bumps: &USDonInitRolesBumps) ->
↪  Result<()> {
    require!(
        matches!(
            role,
            RoleType::MinterRoleUsdon | RoleType::PauserRoleUsdon |
            ↪  RoleType::BurnerRoleUsdon
        ),
        OndoError::InvalidRoleType
    );
    // [...]
}
```

Looking at all other code segments giving roles, there is always another codepath allowing removal of the given roles except for this one.

Recommendation: Consider adding a function `remove_usdon_roles`, allowing the guardian to take away given roles again.

Ondo Finance: Fixed in commit 2f4f7ac4.

Cantina Managed: Fix verified.

3.4.23 Several defined roles are unused in the Solana program

Severity: Low Risk

Context: roles.rs#L22-L51

Description: The Solana program defines multiple `RoleType` variants and seeds that are never referenced outside `roles.rs`, i.e. no instruction uses them for access control or behavior:

- `TokenFactoryRole`.
- `PauserRoleTokenManagerRegistrar`.

- `AdminRoleTokenManagerRegistrar`.
- `PauseTokenRole`.
- `AdminRolePauseToken`.
- `ComplianceOwnerRole`.
- `OwnerIssuanceHoursRole`.

The Solana specification and the Solidity reference design both describe functionality mapped to these roles (e.g. TokenManagerRegistrar pausing/config, compliance owner, issuance-hours owner, token-pause admin), but there is no corresponding implementation on Solana. This creates a specification/implementation gap and can mislead integrators.

Recommendation: It is recommended to decide per role whether it should be implemented or removed.

Ondo Finance: Fixed in commit 1a869cb7. Removes most unused roles, "OWNER_ISSUANCE_HOURS_ROLE" to be used.

Cantina Managed: Fix verified.

3.4.24 Defined but unused error codes indicate missing or incomplete validations

Severity: Low Risk

Context: errors.rs#L4-L97

Description: Several error variants are defined in `errors.rs` but are never used anywhere else in the Solana program. This suggests that some intended validations or safety checks are missing or only partially implemented:

- Swap / pricing related:
  - `InvalidOutputMint`.
  - `SlippageExceeded`.
  - `InvalidMints`.
  - `TokenSwapPaused`.
- Access control / compliance related:
  - `AddressAlreadyInRole`.
  - `BlocklistNotInitialized`.
  - `UserNotWhitelisted`.
- Attestation / signature diagnostics:
  - `InvalidInstructionIndex`.
  - `AttestationSignerEthAddressNotSet`.
  - `PubkeyRecoveryFailed`.
  - `EthAddressRecoveryFailed`.
  - `EthAddressMismatch`.
  - `InvalidSignatureParams`.

Overall, these unused error codes show a mismatch between documented/intended behavior and what is actually enforced on-chain, and they can mislead integrators into assuming certain protections (slippage limits, swap-level pause, blocklist, richer signature diagnostics) exist when they do not.

Recommendation: It is recommended for each unused error to either implement the intended validation or remove the error to reflect the actual behavior.

Ondo Finance: Fixed in commit c247ed8b.

Cantina Managed: Fix verified.

3.4.25 `init_usdon_roles` misses `RoleGranted` event

Severity: Low Risk

Context: usdon_admin_operations.rs#L74

Description: All of the role setting functions emit the `RoleGranted` event.

```
// Emit event for role granted
emit!(RoleGranted {
    role,
    grantee: user,
    granter: self.admin.key(),
});
```

However on the `init_usdon_roles` function the event is not emitted.

Recommendation: We recommend adding an emit.

Ondo Finance: Fixed in commit 2f4f7ac4.

Cantina Managed: Fix verified.

3.4.26 Sanity checker can be initialized with zero `last_price`

Severity: Low Risk

Context: sanity_checker_admin_operations.rs#L77-L85

Description: When setting the last price of the sanity checker using `set_last_price()` it is checked that the price needs to be greater than 0.

```
pub fn set_last_price(&mut self, last_price: u64) -> Result<()> {
    require!(last_price > 0, OndoError::InvalidPrice);
```

However when initializing the sanity checker this is not checked.

```
// Write to the sanity check account
self.sanity_check.set_inner(OracleSanityCheck {
    last_price,
    mint: self.mint.key(),
    allowed_deviation_bps,
    max_time_delay,
    price_last_updated: Clock::get()?.unix_timestamp,
    bump: bumps.sanity_check,
});
```

Recommendation: We recommend adding a check to the initializer to ensure that `last_price > 0`.

Ondo Finance: Fixed in commit 7afcee22.

Cantina Managed: Fix verified.

3.4.27 `mint` is missing token program check

Severity: Low Risk

Context: usdc_swap_context.rs#L29-L34, usdon_swap_context.rs#L28-L33

Description: Both the `USDonSwapContext` as well as the `UsdcSwapContext` include the `mint` account in their context. This account is the gm token that the swap/redeem will be done for.

```
/// The GM Token mint involved in the swap
#[account(
    mut,
    mint::authority = mint_authority,
)]
pub mint: Box<InterfaceAccount<'info, Mint>>,
```

The account constraints however never check the token program of these mints. As a result a incorrect mint could be passed. This is currently mitigated by other restrictions, but it's highly recommended to always check the token program on every mint/token account.

Recommendation: We recommend adding a check for the token program.

Ondo Finance: Fixed in commit 4ce72532.

Cantina Managed: Fix verified.

3.4.28 Users will loose up to 999 lamports of USDon on each USDC redemption

Severity: Low Risk

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

Description: On a USDC redemption, the user is first minted the corresponding value in USDon.

```
mint_to(
    CpiContext::new_with_signer(
        ctx.token_program.to_account_info(),
        MintTo {
            mint: ctx.usdon_mint.to_account_info(),
            to: ctx.user_usdon_token_account.to_account_info(),
            authority: ctx.mint_authority.to_account_info(),
        },
        signer_seeds,
    ),
    mint_amount,
)?;
```

Afterwards a swap will be called that will first noralize the amount to USDC decimals:

```
let normalized_amount_out = normalize_decimals(amount_in, self.usdon_mint.decimals,
↪  usdc_mint.decimals)?;
```

Afterwards it will draw the full `mint_amount` from the user and transfer him `normalized_amount_out` of USDC.

```
transfer_checked(
    CpiContext::new(
        self.token_program.to_account_info(),
        TransferChecked {
            from: self.user_usdon_token_account.to_account_info(),
            mint: self.usdon_mint.to_account_info(),
            to: self.usdon_vault.to_account_info(),
            authority: self.user.to_account_info(),
        },
    ),
    amount_in,
    self.usdon_mint.decimals,
)?;

// Step 2: Transfer USDC tokens from protocol vault to user
// This releases USDC from the protocol's vault to the user's account
if normalized_amount_out != 0 {
    transfer_checked(
        CpiContext::new_with_signer(
            self.spl_token_program
                .as_ref()
                .ok_or(OndoError::TokenProgramNotProvided)?
                .to_account_info(),
            TransferChecked {
                from: self
                    .usdc_vault
                    .as_ref()
                    .ok_or(OndoError::InvalidTokenAccount)?
                    .to_account_info(),
```

```
                mint: usdc_mint.to_account_info(),
                to: self
                    .user_usdc_token_account
                    .as_ref()
                    .ok_or(OndoError::InvalidTokenAccount)?
                    .to_account_info(),
                authority: self.usdon_manager_state.to_account_info(),
            },
            &[&[USDON_MANAGER_STATE_SEED, &[self.usdon_manager_state.bump]]],
        ),
        normalized_amount_out,
        usdc_mint.decimals,
    )?;
}
```

This leads to the user effectively overpaying for USDC. The two tokens should be pegged at a 1:1, but in this case, the user will lose up to 999 lamports of USDon. If, for example, the `mint_amount == 1999`, the conversion would normalize this to `normalized_amount_out == 1`. However, it would draw the full `1999` lamports of USDon from the user, which would be worth `1.999 USDC`. So the user would, in that case, lose the `999` lamports of USDon to the protocol.

Recommendation: We recommend only drawing

```
normalize_decimals(normalized_amount_out, usdc_mint.decimals, self.usdon_mint.decimals)
```

from the USDon vault, which will let the user keep the rounding losses.

Ondo Finance: Fixed in commit 17e9f02d.

Cantina Managed: Fix verified.

3.4.29 Type conversion can lead to unexpected behavior

Severity: Low Risk

Context: capacity.rs#L22

Description: Both the user's and any GM token's `limit_window` can be set by the `ADMIN_ROLE_GMTOKEN_MANAGER` up to `u64::MAX`. The problem is that setting it to any value larger than `i64::MAX` for an account will freeze any interactions with that account.

This is because in `calculate_capacity_used`, which is used by `check_token_rate_limit` and `check_user_rate_limit`, the `limit_window` is cast to an `i64` and `i64::try_from(limit_window)` errors for any value larger than `i64::MAX`:

```
if time_since_last_update >= i64::try_from(limit_window)? {
```

Recommendation: Consider casting `time_since_last_update` (to a `u64`) for the check instead of `limit_window`. This is safe since `time_since_last_update` should never be negative in this context anyways.

Ondo Finance: Fixed in commit 77ccb283.

Cantina Managed: Fix verified.

3.4.30 Signature verification has more restrictions than intended

Severity: Low Risk

Context: token_manager.rs#L227-L237

Description: The current implementation of the signature verification in `verify_secp256k1_ix` is supposed to pass whenever there is "at least one valid secp instruction in the transaction". Right now this is not the case due to two bugs.

- Iterating over 20 instructions:

```
fn verify_secp256k1_ix(
    &self,
    ix_sysvar: &AccountInfo,
    expected_digest32: &[u8; 32],
    expected_eth_address20: [u8; 20],
) -> Result<()> {
    // Iterate through the instructions in the sysvar to find a matching secp256k1
    ↪  instruction
    for i in 0..20 {
        // [...]
    }
    err!(SecpError::MissingOrMismatchedSecpIx)
}
```

`verify_secp256k1_ix` iterates over a maximum of 20 instructions. Since it's technically possible to have transactions with more than 20 instructions in solana, such a transaction would fail even if it's valid and contains a valid secp instruction.

- Only first instruction considered: Looking at `secp_matches`, the function returns an error if the passed instruction does not pass validation.

```
fn secp_matches(
    &self,
    ix: &Instruction,
    digest: &[u8; 32],
    eth_addr: [u8; 20],
) -> Result<bool> {
    // [...]
    require!(!data.is_empty(), SecpError::MalformedSecpIx);
    require!(data[0] == 1, SecpError::WrongSigCount);
    // [...]
    require!(data.len() >= rd + 11, SecpError::MalformedSecpIx);
    // [...]
    require!(msg_len == 32, SecpError::WrongDigestLen);
    require!(msg_off + msg_len <= data.len(), SecpError::MalformedSecpIx);
    require!(eth_off + 20 <= data.len(), SecpError::MalformedSecpIx);
    // [...]
    require!(msg == digest, SecpError::DigestMismatch);
    require!(eth_addr_in_ix == eth_addr, SecpError::AddressMismatch);

    Ok(true)
}
```

This means `verify_secp256k1_ix` will error if the first found secp instruction is not the one it's looking for, even if there is a valid one later on in the instruction list.

```
if self.secp_matches(&ix, expected_digest32, expected_eth_address20)? {
    return Ok(());
}
```

Recommendation: Consider changing the semantics of the signature verification, enforcing that signature instructions must be one index before the instruction they are verified in.

For example, if a transaction contains two `redeem_for_usdon` instructions, one at index 2 and the other one at index 5, then the corresponding signature instructions must be at index 1 and 4 respectively. This would simplify the signature check and allow for bundling of mints/redemptions.

Ondo Finance: Fixed in commit 9f06e4ae.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Role initializing/closing is dependent on mutability of program

Severity: Informational

Context: role_operations.rs#L43-L47, role_operations.rs#L111-L115

Description: Currently the program's upgrade authority is taken as the highest authority in the program since it is the only one capable of giving/taking admin roles.

```
#[account(mut)]
pub admin: Signer<'info>,

// [...]

#[account(
    constraint =
        program_data.upgrade_authority_address == Some(admin.key()) @
        ↪  OndoError::InvalidUser
)]
pub program_data: Account<'info, ProgramData>,
```

The problem with this approach is that in case the decision is made to make the program immutable, this check can never be passed anymore since making a program immutable sets its `upgrade_authority_address` to `None`.

Recommendation: In case the program will definitely never be made immutable, this is fine. If this might be done, however, consider adding an admin role with the same privileges as the upgrade authority in order to have the possibility to manage admin roles even if the program is made immutable.

Ondo Finance: Acknowledged. This is intentional. If a contract is ever planned to be made immutable we will update this. On another note, all roles set by `InitRoles` are only really meant to be set during deployment.

Cantina Managed: Acknowledged.

3.5.2 Tokens with `transfer-allowed == false` can still be transferred

Severity: Informational

Context: token_limit.rs#L36-L37

Description: The `TokenLimit` account is used to constrain mints. It includes the `transfers_allowed` bool, which should disable transfers if set to false.

```
// Whether transfers are allowed for this token
pub transfers_allowed: bool,
```

However, this value is actually never checked anywhere. Thus, transfers for tokens with this set to false will work without any issues.

Recommendation: We recommend disabling transfers for tokens with this value set.

Ondo Finance: Fixed in commit 8cfc3326.

Cantina Managed: Fix verified.

3.5.3 Misleading `is_paused` parameter name for `enable_oracle_price`

Severity: Informational

Context: usdon_manager_admin_operations.rs#L120-L130, lib.rs#L84-L88

Description: The `enable_oracle_price` admin instruction uses a parameter named `is_paused`, but it directly sets the `oracle_price_enabled` flag, which inverts the intuitive meaning of the name and is inconsistent with the rest of the pause naming in the codebase.

In the admin implementation:

```rust
pub fn enable_oracle_price(&mut self, is_paused: bool) -> Result<()> {
    // Set the oracle price enabled state
    self.usdon_manager_state.oracle_price_enabled = is_paused;

    Ok(())
}
```

and exposed from the program:

```rust
pub fn enable_oracle_price(ctx: Context<USDonManagerAdmin>, is_paused: bool) ->
↪  Result<()> {
    ctx.accounts.enable_oracle_price(is_paused)
}
```

Recommendation: It is recommended to rename the parameter to e.g. `oracle_price_enabled: bool` (and update both `lib.rs` and `usdon_manager_admin_operations.rs`), so the call site matches the stored field and avoids inversion/confusion.

Ondo Finance: Fixed in commit 3e16889d.

Cantina Managed: Fix verified.

3.5.4 Incorrect documentation of `PauseGmToken`

Severity: Informational

Context: gm_token_admin_operations.rs#L255

Description: The comment above the `PauseGmToken` context states "Requires `UNPAUSER_ROLE_GMTOKEN` role" however actually the caller needs to hold the `PAUSER_ROLE_GMTOKEN` role.

```rust
/// The Roles account verifying the pauser has PAUSER_ROLE_GMTOKEN
/// # PDA Seeds
/// - PAUSER_ROLE_GMTOKEN
/// - Pauser's address
#[account(
    seeds = [RoleType::PAUSER_ROLE_GMTOKEN, pauser.key().as_ref()],
    bump,
)]
pub roles: Account<'info, Roles>,
```

Recommendation: We recommend adapting the documentation to "Requires `PAUSER_ROLE_GMTOKEN` role".

Ondo Finance: Fixed in commit b9a1ec09.

Cantina Managed: Fix verified.

3.5.5 Incorrect documentation of `MAX_SECONDS_EXPIRATION`

Severity: Informational

Context: constants.rs#L53-L54

Description: The `MAX_SECONDS_EXPIRATION` constant is described as "Maximum allowed attestation expiration time (1 year in seconds)". However, it actually restricts the expiry of the price updates, not the attestations.

Recommendation: We recommend adapting the comment to "Maximum allowed price delay".

Ondo Finance: Fixed in commit b9a1ec09.

Cantina Managed: Fix verified.

3.5.6 Incorrect documentation of `GmTokenManagerAdminGlobalPauser`

Severity: Informational

Context: gm_token_manager_admin_operations.rs#L344-L348

Description: The `GmTokenManagerAdminGlobalPauser` context's functionality is described as "/// Unpause subscriptions/redemptions for all GM Tokens". However, actually, the function handles both pausing and unpausing.

Recommendation: We recommend adapting the comment to "/// Pause/Unpause subscriptions/redemptions for all GM Tokens".

Ondo Finance: Fixed in commit b9a1ec09.

Cantina Managed: Fix verified.

3.5.7 `trading_hours_offset` missing in `initialize_gmtoken_manager` comment

Severity: Informational

Context: gm_token_manager_admin_operations.rs#L49-L66

Description: The NatSpec comment for the `initialize_gmtoken_manager` function is missing a description for the `trading_hours_offset` argument.

```
/// Initialize the GmTokenManagerState account
/// # Arguments
/// * `factory_paused` - Whether the GM Token factory should start in a paused state
/// * `redemptions_paused` - Whether redemptions should start in a paused state
/// * `subscriptions_paused` - Whether subscriptions should start in a paused state
/// * `attestation_signer_secp` - The secp256k1 Ethereum address of the attestation
 ↪  signer (20 bytes)
/// * `bumps` - The PDA bumps for account derivation
/// # Returns
/// * `Result<()>` - Ok if the GmTokenManagerState is successfully initialized, Err
 ↪  otherwise
pub fn initialize_gmtoken_manager(
    &mut self,
    factory_paused: bool,
    redemption_paused: bool,
    minting_paused: bool,
    attestation_signer_secp: [u8; 20],
    trading_hours_offset: i64,
    bumps: &InitializeGmTokenManagerBumps,
) -> Result<()> {
```

Recommendation: We recommend adding a description for the `trading_hours_offset` argument.

Ondo Finance: Fixed in commit b9a1ec09.

Cantina Managed: Fix verified.

3.5.8 Metadata update authority set to program PDA but no update path implemented

Severity: Informational

Context: token_factory.rs#L150-L160

Description: When deploying new mints in `token_factory.rs`, the Token-2022 metadata is initialized with the program's mint authority PDA as both mint authority and update authority.

However, the current codebase does not implement any CPI to update metadata, so in practice metadata is never changed on-chain. The configuration thus sits in an ambiguous state:

- To users, it may look like metadata is immutable.
- To developers, it is technically upgradable by future program changes (since the program holds the update authority PDA), even though no current path exists.

This is not an immediate exploit, but is important for understanding trust and upgrade assumptions around token metadata.

Recommendation: It is recommended to decide explicitly whether metadata should be immutable or upgradable by the program.

Ondo Finance: Fixed in commit 6815abc3.

Cantina Managed: Fix verified.

3.5.9 Unnecessary role checks for `AdminRoleGmtokenManager`

Severity: Informational

Context: token_limit_admin_operations.rs#L68-L71, token_limit_admin_operations.rs#L179-L182

Description: Both the `initialize_token_limit` and the `set_token_limit` functions enforce the following check.

```
require!(
    self.roles.role == RoleType::AdminRoleGmtokenManager,
    OndoError::AddressNotFoundInRole
);
```

However the role being the correct one is already verified based on the seed.

```
#[account(
    seeds = [RoleType::ADMIN_ROLE_GMTOKEN_MANAGER, admin.key().as_ref()],
    bump = roles.bump,
)]
pub roles: Account<'info, Roles>,
```

Recommendation: We recommend removing the unnecessary checks.

Ondo Finance: Fixed in commit 062b5dff.

Cantina Managed: Fix verified.

3.5.10 Incorrect comment in sanity checker

Severity: Informational

Context: sanity_checker_admin_operations.rs#L71

Description: The following check in the sanity check is described as ”Validate last price”.

```
// Validate last price
require!(
    max_time_delay <= MAX_SECONDS_EXPIRATION, // 1 year lifetime in days, to be adjusted
    OndoError::InvalidMaxTimeDelay
);
```

However this actually validates the price delay.

Recommendation: We recommend changing this to ”// Validate price delay”.

Ondo Finance: Fixed in commit 3109a7d8.

Cantina Managed: Fix verified.

3.5.11 Incorrect event emission in `set_token_limit`

Severity: Informational

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

Description: In `set_token_limit`, at the end an event is emitted indicating the newly set values.

```
emit!(RateLimitTokenSet {
    token: self.mint.key(),
    limit: self.token_limit.rate_limit,
    limit_window: self.token_limit.limit_window,
});
```

This is not entirely correct, however, since it is possible that `self.token_limit.rate_limit` or `self.token_limit.limit_window` is set to `None` in which case the values' default values are used instead. Looking at `initialize_token_limit`, it is correctly differentiating between the two cases:

```
emit!(RateLimitTokenSet {
    token: self.mint.key(),
    limit: if self.token_limit.rate_limit.is_some() {
        self.token_limit.rate_limit
    } else {
        self.token_limit.default_user_rate_limit
    },
    limit_window: if self.token_limit.limit_window.is_some() {
        self.token_limit.limit_window
    } else {
        self.token_limit.default_user_limit_window
    },
});
```

Recommendation: Consider changing the event emission in `set_token_limit` to match the one in `initialize_token_limit`.

Ondo Finance: Fixed in commit 4d90e674.

Cantina Managed: Fix verified.

3.5.12 Incorrect documentation of `mint_usdon` and `burn_usdon`

Severity: Informational

Context: lib.rs#L334-L345

Description: Both the `mint_usdon` and `burn_usdon` functions state that they can only be called by either the mint or burn role.

```
/// Mint USDon tokens (admin function)
/// Signer must have the MINTER_ROLE_USDON role
pub fn mint_usdon(ctx: Context<USDonMinter>, amount: u64) -> Result<()> {
    ctx.accounts.mint_usdon(amount, ctx.bumps.mint_authority)
}

/// Burn USDon tokens (admin function)
/// Signer must have the BURNER_ROLE_USDON role
pub fn burn_usdon(ctx: Context<USDonBurner>, amount: u64) -> Result<()> {
    ctx.accounts
        .burn_usdon(amount, ctx.bumps.permanent_delegate)
}
```

However actually they can both also be called by the `ADMIN_ROLE_USDON`.

Recommendation: We recommend adapting the documentation to state that `ADMIN_ROLE_USDON`.

Ondo Finance: Fixed in commit 2f4f7ac4.

Cantina Managed: Fix verified.

3.5.13 Missing access control documentation on `initialize_usdon_manager` function

Severity: Informational

Context: lib.rs#L28-L50

Description: The `initialize_usdon_manager` function is restricted so it can only be called by the `GUARDIAN_USDON`.

```
#[account(
    seeds = [RoleType::GUARDIAN_USDON, admin.key().as_ref()],
    bump = roles.bump,
)]
pub roles: Account<'info, Roles>,
```

However compared to the other functions in `lib.rs` this is not documented.

```
/// Initialize the USDon manager state
///
/// Sets up the manager with the USDon mint, initial price, oracle configuration,
/// and vault addresses for USDC and USDon tokens.
```

Recommendation: We recommend adding the following comment:

```
/// Signer must have the GUARDIAN_USDON role
```

Ondo Finance: Fixed in commit 2f4f7ac4.

Cantina Managed: Fix verified.

3.5.14 Incorrect role comments for sanity checker in `lib.rs`

Severity: Informational

Context: lib.rs#L581-L635

Description: The comments for the sanity checker functions in the `lib.rs` file mention the roles `SETTER_ROLE_SANITY_CHECK`, `ADMIN_ROLE_SANITY_CHECK` and `CONFIGURER_ROLE_SANITY_CHECK`. However actually the roles are called `SETTER_ROLE_ONDO_SANITY_CHECK`, `CONFIGURER_ROLE_ONDO_SANITY_CHECK` and `ADMIN_ROLE_ONDO_SANITY_CHECK`.

Recommendation: We recommend adapting the comments to show the correct roles.

Ondo Finance: Fixed in commit 7afcee22.

Cantina Managed: Fix verified.

3.5.15 `rate_limit_check` should use `PRICE_SCALING_FACTOR`

Severity: Informational

Context: token_manager.rs#L375

Description: `rate_limit_check` scales the amount it uses to check the rate limt using `GM_TOKEN_SCALING_FACTOR`. The functions using `rate_limit_check` (`mint_with_attestation` and `redeem_with_attestation`), however, scale the amount using `PRICE_SCALING_FACTOR`.

This means that in case the two differed, the rate limits and the actual value could deviate from each other. Currently these two are set to the same value though, making this an informational remark.

Recommendation: Consider using the same scaling factor (`PRICE_SCALING_FACTOR`) in `rate_limit_check` and `*_with_attestation` and removing `GM_TOKEN_SCALING_FACTOR` sicne it's unused otherwise anyways.

Ondo Finance: Fixed in commit ebca5863.

Cantina Managed: Fix verified.

3.5.16 `closer` unnecessarily mutable in `CloseAttestationAccount`

Severity: Informational

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

Description: In the `CloseAttestationAccount` context struct, the `closer` is marked as mutable which is not necessary and should be avoided if not needed.

```
pub struct CloseAttestationAccount<'info> {
    /// The user closing the attestation account
    #[account(mut)]
    pub closer: Signer<'info>,
    // [...]
}
```

Recommendation: Consider removing `#[account(mut)]` from the signer (`closer`).

Ondo Finance: Fixed in commit f7c90ce2.

Cantina Managed: Fix verified.