Scanned Code Report

AUDITAGENT

Powered by NETHERMIND SECURITY

AUDITAGENT

Important Notice

This Report has been generated entirely by AI and has not been manually reviewed by Nethermind's security team. It does not constitute a full security audit. Projects may not represent or imply otherwise in any public communication. All findings, observations, and recommendations may contain errors or omissions and must be independently verified by a qualified human reviewer before being acted upon. This Report should be used as a supplementary tool only, alongside professional security practices.

Code Info

Auditor Scan

Scan ID 1	Date November 03, 2026
Organization offchange	Repository offchange-escrow
Branch main	Commit Hash a3d27f70...a26fe836

Contracts in scope

programs/offchange_escrow/src/lib.rs

Code Statistics

Findings 3	Contracts Scanned 1	Lines of Code 717

Findings Summary



Total Findings

- High (0)
- Medium (1)
- Low (1)
- Info (1)
- Best Practices (0)

Code Summary

The `offchange_escrow` program is a Solana-based dark pool escrow protocol built with the Anchor framework for trading SPL and compatible Token-2022 tokens against native SOL. It enables users to place resting buy and sell orders on-chain while keeping their target execution price and payout destination hidden behind a cryptographic SHA-256 commitment. An authorized off-chain matching authority coordinates order discovery and reveals the committed parameters at the time of settlement, ensuring that funds can only move according to the exact terms committed to by the order creator.

Architecture and Mechanics

- **Cryptographic Commitments**: When creating an order, the maker provides a 32-byte SHA-256 hash derived from the order price (denominated in lamports per `1,000,000` token base units), the designated payout public key, and a 32-byte secret salt. During trade execution, these parameters are revealed on-chain and verified against the stored commitment.
- **Escrow Storage**: Sell orders initialize an `Order` Program Derived Address (PDA) alongside a dedicated SPL token `vault` PDA that holds the maker's deposited tokens. Buy orders initialize an `Order` PDA that directly holds the maker's escrowed native SOL lamports.
- **Token Standard Support**: The protocol supports standard SPL tokens and Token-2022 mints. For Token-2022 assets, the program enforces a strict extension whitelist (allowing extensions such as metadata, group pointers, interest-bearing configs, and scaled UI amounts) to ensure that transfer fees or transfer-blocking hooks cannot interfere with escrow accounting.
- **Settlement and Midpoint Crossing**: Trades can be settled either via direct taker fills (`fill_sell` and `fill_buy`), which execute at the resting order's committed price, or via order crossing (`cross`), where two compatible resting orders (`buy_price >= sell_price`) are matched against each other at the exact midpoint of their committed prices. Orders support partial fills, and once an order is completely filled, its escrow accounts are automatically closed and rent is refunded to the maker.
- **Fee Structure and Administration**: A global `Config` PDA tracks the protocol administrator, the authorized matcher public key, the fee recipient wallet, the fee rate in basis points (capped at 100 bps or 1%), and a circuit-breaker pause state. Protocol fees are deducted from the SOL leg of every trade.
- **Unrestricted Cancellations**: Makers retain full custody rights to cancel their open or partially filled orders at any time—even when the protocol is paused or after the order has expired—reclaiming all remaining tokens or SOL along with account rent.

Actors

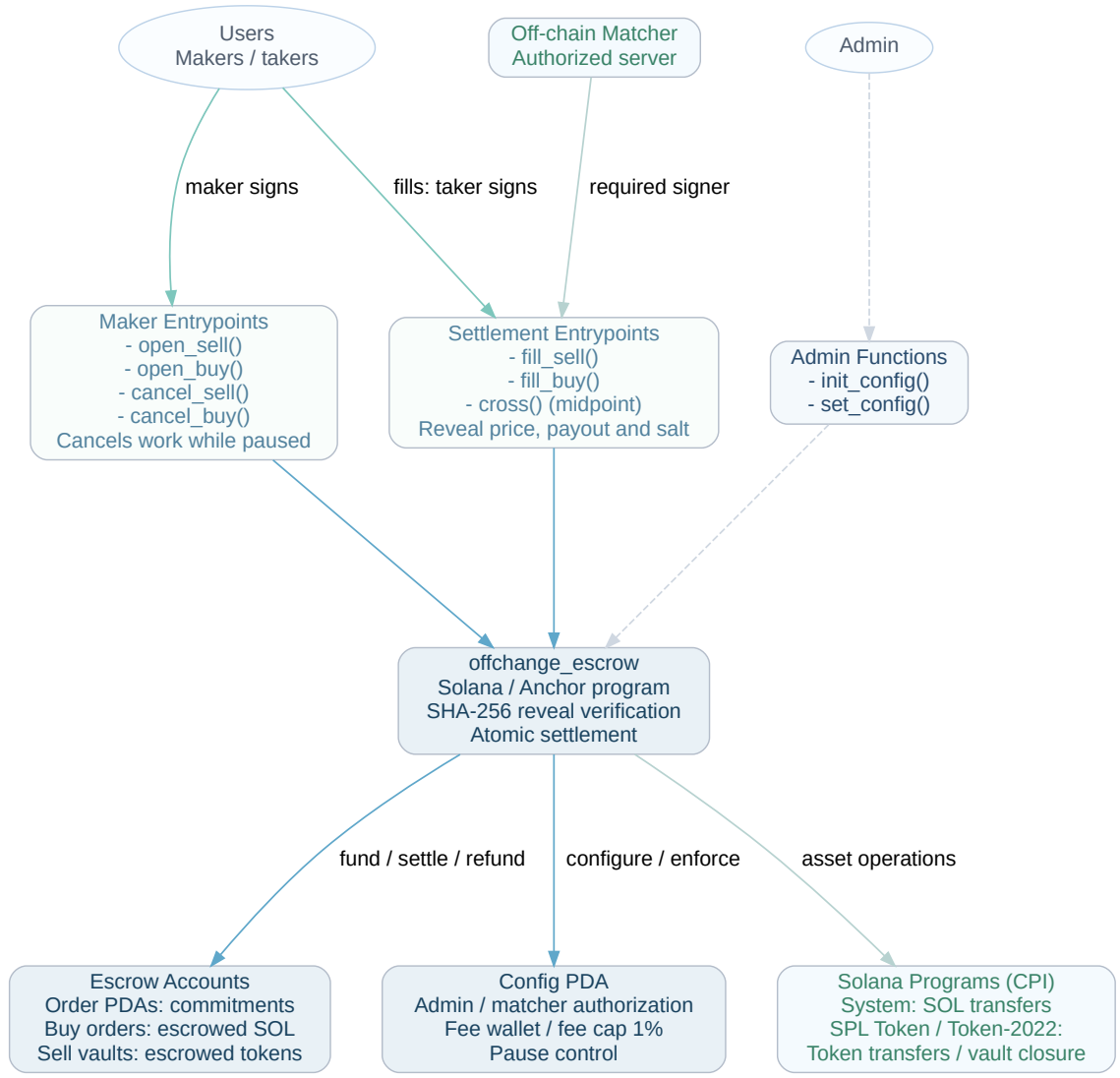
- **Maker**: A trader who creates resting buy or sell orders by locking SOL or tokens in escrow with a hidden price commitment, and who can cancel their orders to reclaim funds and rent.
- **Taker**: A trader who fills a resting buy or sell order by supplying the counter-asset in a transaction co-signed by the matcher.
- **Matcher**: An authorized off-chain server or relayer that holds order preimages, co-signs taker fills, and executes midpoint crosses between compatible resting orders.

Entry Points

- `open_sell`: Allows a **Maker** to initialize a sell order PDA and transfer SPL or Token-2022 tokens into a newly created escrow vault PDA with a price commitment and expiration timestamp.

- `open_buy`: Allows a **Maker** to initialize a buy order PDA and transfer native SOL lamports into the order account with a price commitment and expiration timestamp.
- `cancel_sell`: Allows a **Maker** to cancel their sell order at any time, transferring all remaining tokens from the vault back to the maker and closing both the vault and order accounts to refund rent.
- `cancel_buy`: Allows a **Maker** to cancel their buy order at any time, closing the order PDA and returning all remaining escrowed SOL and rent lamports to the maker.
- `fill_sell`: Allows a **Taker**, co-signed by the **Matcher**, to buy tokens from a resting sell order by revealing the seller's commitment, paying SOL to the seller's payout address and the protocol fee wallet, and receiving tokens from the escrow vault.
- `fill_buy`: Allows a **Taker**, co-signed by the **Matcher**, to sell tokens into a resting buy order by revealing the buyer's commitment, transferring tokens to the buyer's payout token account, and receiving escrowed SOL net of protocol fees.
- `cross`: Allows the **Matcher** to match and settle a resting sell order and a resting buy order at the midpoint of their committed prices, transferring tokens from the sell vault to the buyer's payout token account and transferring SOL from the buy order to the seller's payout wallet and the fee wallet.

Code Diagram



1 of 3 Findings

programs/offchange_escrow/src/lib.rs

`init_config` is permissionless and can be front-run to seize the admin, matcher and fee wallet roles

Medium

Context

`init_config` creates the singleton `Config` PDA (`seeds = [b"config"]`). It writes `admin`, `matcher`, `fee_wallet` and `fee_bps`. Every fill, cross and open reads this config. Only `config.admin` can later change it through `set_config`. The documentation states the deployer/admin is expected to call `init_config` after deployment.

```
// File: programs/offchange_escrow/src/lib.rs
#[derive(Accounts)]
pub struct InitConfig<'info> {
    // @auditagent> any signer can initialize
    #[account(mut)]
    pub admin: Signer<'info>,
    #[account(init, payer = admin, space = 8 + Config::INIT_SPACE, seeds = [b"config"],
bump)]
    pub config: Account<'info, Config>,
```

Root Cause

The `InitConfig` accounts struct only requires an arbitrary `admin: Signer` paying for an `init` account. Nothing binds the signer to the program's upgrade authority or to a hardcoded deployer key. The handler sets `c.admin = ctx.accounts.admin.key()`. The first transaction to reach the cluster becomes the permanent admin. Because the PDA is a singleton with fixed seeds, the legitimate deployer's later `init_config` fails with an already-initialized error.

Impact

An attacker watching for the program deployment can call `init_config` first. They become `admin`, set `matcher` and `fee_wallet` to their own keys, and set `fee_bps` to the 1% maximum. They can also pause the protocol at will. The real off-chain matcher can then no longer co-sign fills or crosses. If the team does not notice and users start trading, every fill pays protocol fees to the attacker. The team cannot recover the config through the program itself and must ship a program upgrade to fix it. Funds already escrowed in orders remain cancellable by their makers, and the attacker cannot alter committed prices or payouts.

Severity Note:
- No order account can be created until Config exists, so the initialization race cannot take escrowed coins or SOL. Makers can always cancel, including while paused, and a fill still has to match the maker’s price and payout commitment.
- Fee diversion is capped at 1% and occurs only if this deployment is used for fills or crosses after initialization was lost. A failed init_config is visible to the deployer before any user funds arrive.
- The seized Config cannot be repaired inside this program. Fixing it requires a program upgrade or a new program id; this source does not show whether the upgrade authority is still held.

2 of 3 Findings

programs/offchange_escrow/src/lib.rs

Emptying `fee_wallet` disables fee-bearing settlements below the recipient rent minimum

Low

Context

`fill_sell` pays fees through `sol_from_signer` and the System Program. `fill_buy` and `cross` instead credit the configured `fee_wallet` through `move_lamports`. The recipient is an unchecked writable account bound only to the configured public key.

```
// File: programs/offchange_escrow/src/lib.rs
if fee > 0 {
    // @auditagent> Fee recipient rent unchecked
    move_lamports(&order_info, &ctx.accounts.fee_wallet.to_account_info(), fee)?;
}
```

Root Cause

All three settlement paths attempt every positive fee payment without accounting for the recipient's rent transition. If a data-empty System account has zero lamports, crediting a positive amount below its rent-exempt minimum produces a prohibited rent-paying account and the transaction fails. This condition can be manufactured by an independently controlled fee collector withdrawing its entire wallet balance; it need not arise only from an initially unfunded configuration.

Impact

A fee collector without administrator or matcher privileges can make ordinary fee-bearing trades fail by leaving `fee_wallet` empty. At the standard 890,880-lamport minimum for a data-empty account, a 10,000-lamport trade with a 1% fee attempts a 100-lamport credit and fails. The failure affects both System Program payments and direct escrow payments. Settlements roll back atomically, so escrow principal is not lost, but matching is degraded and failed transactions can cost fees. Funding the recipient, changing configuration, or a sufficiently large payment restores availability; zero-fee fills are unaffected.

3 of 3 Findings

programs/offchange_escrow/src/lib.rs

`set_config` can irreversibly replace the administrator with an unusable address

Info

Context

`set_config` authenticates the current administrator and immediately overwrites `Config.admin` with an instruction argument. The supplied program has no separate administrator-recovery path.

```
// File: programs/offchange_escrow/src/lib.rs
let c = &mut ctx.accounts.config;
// @auditagent> Successor control is unverified
c.admin = admin;
c.matcher = matcher;
c.fee_wallet = fee_wallet;
c.fee_bps = fee_bps;
c.paused = paused;
```

Root Cause

The successor does not sign or accept the transfer, and no pending-administrator state exists. A current administrator can therefore accidentally install an address that cannot authorize future instructions, such as `Pubkey::default()`. The same call can set `paused = true`, leaving a configuration that nobody can subsequently unpause through `set_config`.

Impact

An erroneous authority rotation can permanently remove configuration control. If the protocol is paused, new orders and settlements remain unavailable; otherwise, operations can continue only under the frozen configuration. Existing owners retain cancellation access. This is an administrator-error hazard rather than an unprivileged takeover.

Disclaimer

No Guarantee of Accuracy or Completeness. Kindly note, no guarantee is being given as to the accuracy and/or completeness of any of the outputs the AuditAgent may generate, including without limitation this Report. The results set out in this Report may not be complete nor inclusive of all vulnerabilities. The AuditAgent is provided on an 'as is' basis, without warranties or conditions of any kind, either express or implied, including without limitation as to the outputs of the code scan and the security of any smart contract verified using the AuditAgent.

This Report is not a security audit. This Report has been generated automatically by AuditAgent, an AI-powered automated code scanning tool. It does not constitute, and must not be represented or relied upon as constituting, a full or comprehensive security audit conducted by Nethermind or any of its personnel. A formal security audit involves manual review by experienced human auditors and is a materially different service. If you require a full security audit, please contact Nethermind's security audit team at auditagent@nethermind.io

Use of Nethermind's name. "Nethermind" is a trade mark of Demerzel Solutions Limited. Use of this Report does not authorise you to represent that your code or smart contract has been "audited by Nethermind" or to use any similar phrase. Any such representation would be false, misleading, and an infringement of Nethermind's intellectual property rights.]
No Endorsement or Security Guarantee. Blockchain technology remains under development and is subject to unknown risks and flaws. This Report does not indicate the endorsement of any particular project or team, nor guarantee its security. Neither you nor any third party should rely on this Report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset.

Disclaimer. To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this Report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.