Stonewall
Smart Contract Security

# Solana Escrow/Swap Security Audit Report

**Auditor:** Stonewall Security **Date:** January 2026 **Version:** 1.0 **Repository:** rustjesty/solana-escrow-smart-contract **Language:** Rust (Anchor Framework) **Chain:** Solana

## Executive Summary

This audit reviews a Solana escrow program implementing atomic token swaps. Users can create offers to exchange Token A for Token B, and other users can accept these offers to complete the swap trustlessly.

### Findings Summary

| Severity | Count |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 2 |
| Low | 3 |
| Informational | 2 |

## Scope

| File | SLOC |
|---|---|
| lib.rs | ~50 |
| instructions/make_offer.rs | ~80 |

| | |
|---|---|
| instructions/take_offer.rs | ~100 |
| state/offer.rs | ~20 |

### Features Reviewed

- Make offer (deposit tokens to vault)
- Take offer (complete swap)
- Vault management with PDAs
- Token transfer operations

---

## Findings

### [M-01] No Offer Expiration Mechanism

**Severity:** Medium **Status:** Open

**Description:** The `Offer` struct doesn't include an expiration timestamp. Offers remain valid indefinitely until taken or cancelled.

**Current Offer State:**

```
pub struct Offer {
    pub id: u64,
    pub maker: Pubkey,
    pub token_mint_a: Pubkey,
    pub token_mint_b: Pubkey,
    pub token_b_wanted_amount: u64,
    pub bump: u8,
    // Missing: pub expiration: i64,
}
```

**Impact:**

- Stale offers at unfavorable prices can be taken
- Maker's tokens locked indefinitely if no one takes offer
- No way to auto-expire offers

**Recommendation:** Add expiration field and validation:

```rust
pub struct Offer {
    // ... existing fields ...
    pub expiration: i64,  // Unix timestamp, 0 = no expiration
}

// In take_offer:
if offer.expiration != 0 && Clock::get()?.unix_timestamp > offer.expiration {
    return Err(ErrorCode::OfferExpired.into());
}
```

---

## [M-02] Missing Cancel Offer Functionality

**Severity:** Medium **Status:** Open

**Description:** The contract has `make_offer` and `take_offer` but no `cancel_offer` instruction. Makers cannot retrieve their tokens if they change their mind.

**Impact:**

- Makers' tokens permanently locked if offer not taken
- No way to update offer terms
- Poor user experience

**Recommendation:** Add cancel_offer instruction:

```rust
pub fn cancel_offer(ctx: Context<CancelOffer>) -> Result<()> {
    // Verify maker is signer
    // Transfer tokens from vault back to maker
    // Close vault and offer accounts
    Ok(())
}
```

---

## [L-01] Potential Front-Running on Take Offer

**Severity:** Low **Status:** Acknowledged

**Description:** When a user submits `take_offer`, another user could front-run by taking the same offer first.

**Impact:**

- User's transaction fails after paying gas
- Minor inconvenience, no fund loss

**Mitigation Present:** This is inherent to blockchain systems. The offer is deleted atomically, so no double-spend is possible.

---

### [L-02] No Partial Fill Support

**Severity:** Low **Status:** Design Choice

**Description:** Offers must be taken in full. There's no partial fill functionality.

**Impact:**

- Less flexible trading
- Large offers may be harder to fill

**Recommendation:** Consider adding partial fill support for larger offers.

---

### [L-03] init_if_needed Could Allow Account Confusion

**Severity:** Low **Status:** Open

**Description:** The `take_offer` instruction uses `init_if_needed` for taker's token accounts:

```
#[account(
    init_if_needed,
    payer = taker,
    associated_token::mint = token_mint_a,
    associated_token::authority = taker,
)]
pub taker_token_account_a: Box<InterfaceAccount<'info, TokenAccount>>,
```

**Impact:**

- Taker pays for account creation if it doesn't exist
- Generally safe with proper constraints, but adds complexity

**Recommendation:** Consider requiring token accounts to exist beforehand, or clearly document the rent cost to users.

---

### [I-01] Good Use of Token-2022 Interface

**Severity:** Informational

**Description:** The contract uses `TokenInterface` instead of the legacy `Token` program, supporting both SPL Token and Token-2022.

```
pub token_program: Interface<'info, TokenInterface>,
```

This is a positive design choice for forward compatibility.

---

### [I-02] Consider Adding Offer Indexing

**Severity:** Informational

**Description:** Offers use a user-provided `id` field. Consider using a global counter or more structured indexing for easier off-chain querying.

---

## Security Features (Positive Findings)

1. **Proper PDA Usage**: Vault is a PDA owned by the offer account

   ```
   seeds = [b"offer", maker.key().as_ref(), offer.id.to_le_bytes().as_ref()]
   ```

2. **has_one Constraints**: Validates offer fields match provided accounts

   ```
   #[account(
       has_one = maker,
       has_one = token_mint_a,
       has_one = token_mint_b,
   )]
   ```

3. **Atomic Execution**: All transfers happen in single transaction

4. **Proper Account Closing**: Vault and offer accounts closed after completion

5. **Signer Seeds for CPI**: Correct use of PDA signer seeds for vault transfers

## Code Quality

The code follows Anchor best practices:

- Modular structure (separate instruction files)
- Proper use of Box for large accounts
- Clean separation of concerns
- Good use of helper functions ( `transfer_tokens` )

## Conclusion

This is a well-structured escrow contract suitable for token swaps. The main improvements needed are:

1. Add offer expiration
2. Add cancel offer functionality
3. Consider partial fills for better UX

**Overall Assessment:** Low Risk (Missing Features)

---

**Stonewall Security** *Building Stronger Smart Contracts*