

Solana Foundation: DvP Program

Security Review

Cantina Managed review by:
R0bert, Lead Security Researcher
Sujith S, Lead Security Researcher

June 10, 2026

Contents

1 Introduction **2**
1.1 About Cantina . . . 2
1.2 Disclaimer . . . 2
1.3 Risk assessment . . . 2
1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
2.1 Scope . . . 3

3 Findings **4**
3.1 Medium Risk . . . 4
3.1.1 Closed unfunded mint can block funded-leg recovery . . . 4
3.1.2 Memo-required ATAs can block DvP recovery . . . 4
3.1.3 Post-expiry recovery is all-or-nothing . . . 5
3.1.4 Nonce-reusable PDA address after close enables stale-deposit capture across DvP instances . . . 5
3.2 Low Risk . . . 7
3.2.1 `NonTransferable` mints can brick funded DvPs . . . 7
3.2.2 Unsynced WSOL lamports bypass refunds . . . 7
3.2.3 Dust surplus can block settlement when refund ATAs are absent . . . 8
3.2.4 Destination-dependent `TransferHook` extras can block surplus settlement . . . 8
3.2.5 Permissionless first-come `CreateDvp` permits slot squatting with attacker-chosen terms . . . 9
3.2.6 On-chain `Clock.unix_timestamp` drift breaks wall-clock assumptions in expiry, settlement-window, and reclaim gating . . . 10
3.3 Informational . . . 11
3.3.1 Closeable mints can bypass the extension deny-list . . . 11
3.3.2 Mint authorities can freeze or claw back escrows . . . 11
3.3.3 Closed-account lamports go to terminal signers . . . 12
3.3.4 Confidential transfer fee denial is undocumented . . . 12
3.3.5 `TransferHook` account cap can brick terminal instructions . . . 13
3.3.6 Transfer hooks can receive caller signer privileges . . . 13
3.3.7 README omits Token-2022 funding requirements . . . 14
3.3.8 SBF build leaves deploy keypairs that do not match declared program IDs . . . 14
3.3.9 IDL does not describe `TransferHook` extra accounts . . . 15
3.3.10 Unbounded `expiry_timestamp` permits indefinite escrow lifetime and permanent rent lock-up . . . 16
3.3.11 Settlement window may be configured too narrow to be operationally usable . . . 16

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

From May 21st to May 28th the Cantina team conducted a review of solana-private-channels on commit hash 58ec0f9b. The team identified a total of **21** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	4	4	0
Low Risk	6	3	3
Gas Optimizations	0	0	0
Informational	11	5	6
Total	**21**	**12**	**9**

2.1 Scope

The security review had the following components in scope for solana-private-channels on commit hash 58ec0f9b:

```
dvp-swap-program
├── clients
│   └── rust
│       ├── Cargo.toml
│       └── src
│           └── lib.rs
└── program
    ├── build.rs
    ├── Cargo.toml
    └── src
        ├── discriminator.rs
        ├── entrypoint.rs
        ├── error.rs
        ├── instructions.rs
        ├── lib.rs
        ├── processor
        │   ├── cancel_dvp.rs
        │   ├── create_dvp.rs
        │   ├── mod.rs
        │   ├── reclaim_dvp.rs
        │   ├── reject_dvp.rs
        │   ├── settle_dvp.rs
        │   └── shared
        │       ├── account_check.rs
        │       ├── mod.rs
        │       ├── pda_utils.rs
        │       ├── token_utils.rs
        │       └── utils.rs
        └── state
            ├── mod.rs
            └── swap_dvp.rs
```

3 Findings

3.1 Medium Risk

3.1.1 Closed unfunded mint can block funded-leg recovery

Severity: Medium Risk

Context: cancel_dvp.rs#L84-L89

Description: `CancelDvp` and `RejectDvp` require both mint accounts to still be owned by their recorded token programs before they refund either leg. This check runs even when one leg has a zero escrow balance and no transfer for that mint will be attempted.

Token-2022 mints can include `MintCloseAuthority`. If a DvP is created with a closeable mint on leg A, leg A remains unfunded and leg B is funded, the leg A mint can be closed while the DvP is open as long as its supply is zero. After that, `CancelDvp` and `RejectDvp` fail at the mint-owner check for leg A before refunding leg B. If the DvP has expired, `ReclaimDvp` is also unavailable, so the funded leg can remain locked until the mint account is restored at the same address.

The same issue exists symmetrically for leg B. The root cause is that terminal refund instructions require live mint accounts for unfunded legs even though closing an empty escrow ATA does not need the mint account.

Recommendation: Store each leg's token program in `SwapDvp` at creation and relax terminal refund validation for zero-balance legs. `CancelDvp` and `RejectDvp` should be able to close an empty escrow ATA and refund the opposite funded leg even if the empty leg's mint account was closed. Alternatively, reject mints with `MintCloseAuthority` at `CreateDvp`.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. `SwapDvp` now stores `token_program_a` and `token_program_b` at creation and the shared `CancelDvp` / `RejectDvp` refund path validates the supplied token programs against those stored values rather than requiring the mint accounts to still be owned by those programs. Transfers are skipped for zero-balance legs, so a closed mint on the empty side no longer prevents refunding and closing the funded side. Regression tests cover both directions with a closed unfunded mint.

3.1.2 Memo-required ATAs can block DvP recovery

Severity: Medium Risk

Context: cancel_dvp.rs#L111-L162

Description: `CancelDvp` verifies only that each refund destination is the canonical ATA, then sends `TransferChecked` to that account when the escrow balance is non-zero. `RejectDvp`, `ReclaimDvp` and the settlement/surplus transfers use the same pattern.

This misses a Token-2022 account-level condition: an ATA can enable the `MemoTransfer` extension and require a memo before incoming transfers. The swap program does not invoke the Memo program before its token CPIs and a caller cannot insert a memo inside the program's CPI sequence. Therefore a party can make their own canonical ATA reject incoming refunds or settlement proceeds.

The failure is easiest to see as a two-box escrow. User A funds one escrow box and user B funds the other. If the trade is aborted, the program must return each box to its original party. If user B enables ”memo required” on their refund ATA, the DvP's attempt to return user B's tokens fails because the internal token transfer is not immediately preceded by a Memo CPI. Since the refund instruction is atomic, the entire instruction reverts and user A's refund is rolled back too.

This allows locking of funds and griefing. A counterparty can make its own destination ATA reject incoming refunds or settlement proceeds, causing `CancelDvp`, `RejectDvp`, `ReclaimDvp` or `SettleDvp` to

fail. The worst case is after expiry: `ReclaimDvp` is unavailable, so users must rely on `CancelDvp` or `RejectDvp`, which refund both sides together. If one side's memo-required ATA rejects its refund, the other side's otherwise healthy funded leg can remain locked until the memo-requiring party disables the extension or the program supports Memo CPIs.

Recommendation: Support memo-required Token-2022 destinations in the program. Before each `TransferChecked`, inspect the destination token account for `MemoTransfer`. When a memo is required, invoke the Memo program immediately before the token CPI. Add the Memo program as an explicit account for terminal instructions.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. All terminal transfer paths now pass a `memo_program` account into `transfer_checked_cpi`, which inspects Token-2022 destinations for `MemoTransfer` and invokes the canonical Memo program immediately before the token CPI when required. The regression suite covers memo-required destinations for `CancelDvp`, `RejectDvp`, `ReclaimDvp` and `SettleDvp`, including surplus refunds.

3.1.3 Post-expiry recovery is all-or-nothing

Severity: Medium Risk

Context: reclaim_dvp.rs#L100-L103

Description: `ReclaimDvp` is the only instruction that drains one leg without touching the other leg, but it rejects once `now > expiry_timestamp`. After expiry, users must rely on `CancelDvp` or `RejectDvp`.

Those terminal instructions are coupled across both legs. They validate both mints, verify both escrow and refund ATAs, snapshot both balances, attempt every nonzero refund, then close both escrows and the DvP PDA. If either leg cannot be transferred, the entire instruction reverts. Consequently, a problem with one mint, destination account, or TransferHook can lock the other leg after expiry even when the other leg is healthy and independently transferable.

Several concrete instances are tracked separately, including non-transferable mints, memo-required refund ATAs, closed unfunded mints and hook-program vetoes. This issue is the shared architectural root cause: after expiry there is no independent per-leg recovery instruction.

Recommendation: Allow each party to reclaim their own leg after expiry without touching the other leg. Consider removing the expiry restriction from `ReclaimDvp` or add a new post-expiry per-leg reclaim instruction. If a per-leg recovery can happen after expiry, add a explicit terminal state so settlement cannot later become possible after one party has recovered their leg. Close the DvP PDA only once both escrows are empty.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. `ReclaimDvp` no longer has an expiry gate and only validates the signer-selected leg, so a healthy leg can be drained independently after expiry without touching the other mint, destination, or hook path. `SettleDvp` remains expiry-gated, so post-expiry reclaim does not enable one-sided settlement. The integration test `test_reclaim_dvp_post_expiry_recovers_leg` exercises the fixed path.

3.1.4 Nonce-reusable PDA address after close enables stale-deposit capture across DvP instances

Severity: Medium Risk

Context: create_dvp.rs#L52

Description: The `SwapDvp` PDA is derived from an identity-only seed set [b"dvp", settlement_authority, user_a, user_b, mint_a, mint_b, nonce.to_le_bytes(), bump]. Economic terms (amount_a, amount_b, expiry_-timestamp, earliest_settlement_timestamp) are deliberately not seeds, and the on-chain record carries no per-instantiation discriminator (no creation slot, no version, no init timestamp).

`SettleDvp` / `CancelDvp` / `RejectDvp` fully close the PDA: lamports are zeroed and `swap_dvp_info.close()` reassigns the account to the system program. After close, `verify_system_account(swap_dvp_info, true)` succeeds again at the same address.

Because `CreateDvp` is permissionless only the payer signs; `user_a`, `user_b`, and `settlement_authority` are never required to sign. Hence any party can re-instantiate a closed DvP at the same address with arbitrarily different amount_a/amount_b/expiry by reusing the original nonce.

The escrow ATAs, derived from (PDA, mint, token_program), also land at the same addresses they had under the previous instance.

Combined with funding being a raw, unauthenticated SPL Transfer to the deterministic escrow address with no link to a specific instance (only the escrow's balance matters at Settle/Cancel/Reject time), this address-recycling primitive produces several possible attacks:

1. **Stale-deposit capture (primary impact).** An adversarial counterparty or settlement authority closes a DvP (via Reject/Cancel) and immediately recreates the same (identity, nonce) PDA with predatory terms (e.g. amount_b = 1). Any funding transfer that the victim prepared, queued, retried, or batched against the public escrow address now lands in the new instance, which Settle then drains under the attacker's amounts. The TOCTOU window is the gap between "validate terms" and "send raw SPL transfer" is fundamental to the raw-transfer funding model.
2. **Indexer / custodian state confusion.** Off-chain systems that key trades by PDA address see two distinct logical trades collapsed into one identifier, distinguishable only by slot ordering. Any UI/integration that tells a depositor "trade X lives at address P, send tokens there" off stale state directly enables attack (1).

The existing test `test_two_dvps_same_parties_different_nonces_are_isolated` (`tests/integration-tests/src/test_settle_dvp/mod.rs:471`) only covers different nonces in parallel and there is no test or on-chain check that prevents same-nonce reuse after close.

The comment in `instructions.rs:83-84` frames nonce as "disambiguates DvPs sharing all other seeds," which implicitly assumes nonces are monotonic and unique-for-all-time, but the program does not enforce this and provides no mechanism for clients to discover that an address has been previously used.

Recommendation: The optimal fix is to ensure that an (identity, nonce) PDA address corresponds to exactly one trade instance, forever.

Apply one of:

1. Tombstone used nonces. Maintain a small NonceTombstone PDA per (identity-tuple, nonce) that is never closed by `Settle` / `Cancel` / `Reject`. Create rejects when the tombstone exists.
2. Or Add a per-instance discriminator to the seeds. Include the creation slot (from Clock::get()?.slot) or a program-wide monotonic counter (held in a singleton PDA bumped atomically inside Create) in the PDA seed array and store it on the record. After this change, "same nonce" no longer means "same address" and a recreated DvP lives at a fresh PDA.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. `CreateDvp` now requires and creates a per-DvP nonce tombstone PDA derived from the `SwapDvp` address, and terminal instructions close the trade without closing that tombstone. Re-creating the same `(settlement_authority, users, mints, nonce)` PDA after close fails with `NonceAlreadyUsed`, which blocks stale-deposit capture at the recycled escrow address. The regression test `test_create_dvp_rejects_reused_nonce_after_close` covers the bypass attempt with changed terms.

3.2 Low Risk

3.2.1 `NonTransferable` mints can brick funded DvPs

Severity: Low Risk

Context: token_utils.rs#L90-L138

Description: `validate_mint_extensions` blocks amount-mutating Token-2022 mint extensions but it explicitly allows `NonTransferable`. All terminal instructions recover funds by transferring tokens out of the escrow ATA before closing it.

Token-2022 rejects transfers from accounts associated with a non-transferable mint. If a positive balance reaches a DvP escrow for such a mint, the swap program cannot settle, refund, reclaim or close that leg. This can happen when a mint authority mints directly to the escrow ATA after `CreateDvp`. If the opposite leg is funded, that counterparty's funds are also stuck because `CancelDvp` and `RejectDvp` attempt to drain both escrows atomically.

The comment says unsupported extensions should fail and remain recoverable once the blocking condition lifts. That does not hold for `NonTransferable`, since the transfer restriction is the permanent purpose of the mint extension.

Recommendation: Reject Token-2022 mints with the `NonTransferable` extension at `CreateDvp`. If non-transferable assets must be supported, the design needs a separate issuer-mediated unwind mechanism because the current PDA-owned escrow cannot transfer or burn those tokens.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. `validate_mint_extensions` now rejects Token-2022 mints with `NonTransferable` during `CreateDvp` and the integration test `test_create_rejects_non_transferable_on_mint_a` verifies the rejection. Under the current create-time path, a non-transferable mint cannot be accepted into a new DvP.

3.2.2 Unsynced WSOL lamports bypass refunds

Severity: Low Risk

Context: token_utils.rs#L59-L71

Description: `get_token_account_balance` reads only the SPL token account `amount` field. For native SOL token accounts, lamports can be transferred directly into the account without first calling `SyncNative`, so those lamports are not reflected in `amount`.

`ReclaimDvp`, `CancelDvp`, `RejectDvp` and `SettleDvp` decide how much to refund or deliver from this token amount. Terminal instructions then close the escrow token account to the settlement authority or rejecting signer. Consequently, unsynced WSOL lamports can skip the depositor refund transfer and be paid to the close recipient instead.

This does not affect normal SPL token deposits or correctly synced WSOL transfers. It affects native SOL legs when a depositor or integration funds the escrow with a system transfer and forgets to sync the token account before attempting recovery or settlement.

Recommendation: Either reject the native mint at `CreateDvp` or explicitly support native SOL accounts by detecting native token accounts and calling `SyncNative` before reading balances.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. `get_token_account_balance` now detects native token accounts and invokes `SyncNative` before re-reading the token `amount` and all terminal paths use this helper before deciding transfer/refund amounts. The regression test `test_reject_dvp_syncs_native_escrow_before_refund` confirms unsynced WSOL lamports are refunded to the depositor rather than swept to the close recipient.

3.2.3 Dust surplus can block settlement when refund ATAs are absent

Severity: Low Risk

Context: settle_dvp.rs#L141-L158

Description: `SettleDvp` sends each agreed leg amount to the counterparty. If either escrow holds more than the agreed amount, it treats the extra balance as a surplus and sends that surplus back to the original depositor's canonical ATA for that mint.

The program only checks that the supplied surplus refund address is the right canonical ATA. It does not create the ATA if it is missing. This is fragile because the depositor may have funded the escrow from a custodian or another non-canonical token account and may never have created their own canonical ATA.

Any holder of the leg mint can send a tiny extra amount into the escrow. That turns an otherwise exact settlement into a settlement that must also execute a surplus refund. If the depositor's canonical refund ATA is missing, the surplus `TransferChecked` CPI fails because the destination account is uninitialized. Since `SettleDvp` is atomic, that failed surplus refund reverts the whole settlement, even though both agreed trade amounts were funded.

This is not a permanent fund loss because the missing ATA can usually be created and the settlement retried. The practical impact is a timing grief. If the dust is added close to expiry, the parties may lose the ability to settle the trade before the deadline and may be forced to cancel or reject instead.

Recommendation: Treat surplus refund ATAs as required settlement accounts, not optional edge-case accounts. Clients should create all four user-side ATAs before calling `SettleDvp`: the two ATAs that receive the agreed trade amounts and the two ATAs that may receive surplus refunds.

For a stronger on-chain fix, `SettleDvp` can create missing surplus refund ATAs before transferring, using the ATA and system program accounts. Another option is to separate exact-amount settlement from surplus recovery so a missing surplus refund account cannot block delivery of the agreed trade amounts.

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.2.4 Destination-dependent `TransferHook` extras can block surplus settlement

Severity: Low Risk

Context: settle_dvp.rs#L211-L246

Description: `SettleDvp` uses one transfer-hook extras slice per leg. That same slice is first used for the main delivery transfer and then reused for the surplus refund transfer when the escrow balance is above the agreed amount.

For Token-2022 hooks, `ExtraAccountMetaList` entries can resolve required accounts from the transfer instruction's destination account or amount. The main delivery transfer and the surplus refund transfer have different destinations and, usually, different amounts. Therefore hook extras that are correct for the main transfer can be incorrect for the surplus refund.

Any holder of the leg mint can over-deposit dust into the escrow. If the hook's extra accounts depend on destination or amount, `SettleDvp` then reverts on the surplus refund CPI even though the agreed trade amount is fully funded. The affected party can usually recover by reclaiming before expiry and re-funding exactly the agreed amount or by canceling/rejecting the DvP, but settlement can be blocked while the

surplus remains. If the dust is added close to expiry, the intended DvP may no longer be settleable before the deadline.

Recommendation: Do not reuse one hook extras slice for both delivery and surplus refund transfers. Either accept separate hook extras for each transfer destination, split settlement into an exact-balance path plus a separate refund path or reject surplus settlement for hook-bearing mints whose `ExtraAccountMetaList` depends on destination or amount. Client-side builders should warn when hook-bearing escrows have surplus balances and should avoid funding above the exact agreed amount.

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.2.5 Permissionless first-come `CreateDvp` permits slot squatting with attacker-chosen terms

Severity: Low Risk

Context: create_dvp.rs#L80

Description: The SwapDvp PDA is derived from an identity-only seed set:

```
["dvp", settlement_authority, user_a, user_b, mint_a, mint_b, nonce]
```

The economic terms of the trade: `amount_a`, `amount_b`, `expiry_timestamp`, and `earliest_settlement_timestamp` are persisted in the account's data but are not part of the seeds, so they do not influence the trade's address.

`CreateDvp` is permissionless: the only signature check is `verify_signer(payer_info, true)`, and the payer is not required to be `user_a`, `user_b`, or `settlement_authority`.

Consequently, any third party can pre-create ("squat") the exact slot a pair of counterparties intends to use and populate it with arbitrary terms (any non-zero amounts, any future expiry, any `earliest <= expiry`). Because `user_a` / `user_b` do not sign `CreateDvp`, the on-chain record is not evidence that the named parties agreed to those terms.

This behavior could lead to the following issues:

1. **Denial of service.** The squat occupies the intended (settlement_authority, user_a, user_b, mint_a, mint_b, nonce) address, so the legitimate CreateDvp fails. If the nonce is predictable, the attacker can re-squat immediately after each recovery, producing a persistent DoS on that counterparty/mint pair. Submission to the channel is not access-controlled at the application layer, so the attacker set is broad.
2. **Terms-mismatch footgun.** Escrow ATA addresses derive only from (PDA, mint) and never from the terms. So a client that funds the derivable escrow without reading the stored `SwapDvp` could deposit against terms it never agreed to.

Recommendation:

1. Unpredictable nonce. Require clients to generate a cryptographically random 64-bit nonce per trade so slots cannot be pre-squatted or re-squatted in a loop.
2. Validate stored terms before acting. Mandate that funders read and verify the on-chain SwapDvp (amounts, expiry, earliest, parties, authority) against the agreed deal before depositing, and that the `settlement_authority` re-validate before Settle. Document explicitly that a SwapDvp record may have been created by any third party and its existence is not proof of mutual agreement. Surface this in the client SDK (e.g., a createAndVerify helper) rather than leaving it to integrators.

Optional on-chain hardening (each with a tradeoff; adopt only if stronger guarantees are wanted):

3. Bind terms to the address by including a commitment (e.g., hash(`amount_a` ▯ `amount_b` ▯ `expiry` ▯ `earliest`)) in the PDA seeds, so a record at the expected address provably carries the expected terms. Tradeoff: the client must know all terms to derive the address (minor).

4. Require `user_a` and `user_b` to co-sign `CreateDvp` so the stored terms are authenticated as agreed. Tradeoff: breaks the ”permissionless create / fund via raw SPL transfer / no custom custodian integration” property - likely undesirable for this product.

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.2.6 On-chain `Clock.unix_timestamp` drift breaks wall-clock assumptions in expiry, settlement-window, and reclaim gating

Severity: Low Risk

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

Description: The DvP swap program enforces all time-based invariants against Solana's `Clock::get()?.unix_timestamp`, which is a stake-weighted median of validator vote timestamps not real wall-clock time.

In normal cluster operation this value lags real time by ~10–30 seconds, and during validator catch-up it can also step forward in larger jumps as the median advances.

The README and instruction docs present `expiry_timestamp` and `earliest_settlement_timestamp` as straightforward time cutoffs, with no disclosure that the cutoff is measured against cluster time rather than the wall-clock seconds that off-chain participants reason in. The mismatch surfaces four issues across the four time-gated code paths.

1. **Settlement-authority free option during clock lag.** `settle_dvp.rs` accepts Settle while `cluster_now <= dvp.expiry_timestamp`. Because the cluster clock trails real time, the on-chain Settle window remains open for tens of seconds after the real wall-clock `expiry_timestamp` has passed.
2. **Reclaim can be bricked by a forward clock jump.** `reclaim_dvp.rs` rejects Reclaim once `cluster_now > dvp.expiry_timestamp`. When the cluster median catches up in a step, a depositor whose tx lands in a slot where the clock has just jumped past `expiry_timestamp` while real wall-clock time is still pre-expiry gets DvpExpired.
3. **earliest_settlement_timestamp is not a hard time-lock.** `settle_dvp.rs` gates `settlement on cluster_now >= earliest`. A forward cluster-clock jump can flip this predicate from false to true seconds before real wall-clock earliest, allowing settlement before the wall-clock-defined window opens.
4. **Create-time expiry_timestamp > now check has the same lag hole.** `create_dvp.rs` rejects Create when `expiry_timestamp <= cluster_now`. With cluster lag of N seconds, a permissionless payer (Create is permissionless by design) can successfully create a DvP whose `expiry_timestamp` is already in the past in wall-clock terms and the trade is born wall-clock-expired.

In all four cases the code does exactly what its comments claim, there is no logic bug. The exposure is that the protocol treats `Clock.unix_timestamp` as a synonym for ”now,” while every off-chain participant, integrator, UI, and policy engine reasons in wall-clock seconds.

Recommendation: This is best treated as a documentation-and-integration issue first, with an optional protocol-level mitigation for the settlement-authority option problem if the threat model demands it.

1. Document the clock model explicitly in dvp-swap-program/README.md. Add a section describing that `expiry_timestamp` and `earliest_settlement_timestamp` are evaluated against `Clock.unix_timestamp` (cluster-vote median, lagging real time by ~10–30s in normal operation and capable of forward jumps), not wall-clock time.
2. Establish a recommended drift budget (e.g. 60s) and make the protocol be tolerant to the drift budget.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. The README now explicitly states that `expiry_timestamp` and `earliest_settlement_timestamp` are evaluated against `Clock::unix_timestamp`, describes the cluster-time drift/jump model and recommends budgeting about 60 seconds away from time boundaries. This removes the prior wall-clock documentation mismatch while leaving the on-chain time source unchanged.

3.3 Informational

3.3.1 Closeable mints can bypass the extension deny-list

Severity: Informational

Context: token_utils.rs#L126-L135

Description: `CreateDvp` rejects Token-2022 mints with amount-mutating extensions, but it accepts mints with `MintCloseAuthority`. A close authority can close a zero-supply mint after `CreateDvp`, then recreate the same mint address with a different token program or with a previously denied extension such as `TransferFeeConfig`.

Terminal instructions do not store or re-check the token program and extension set accepted at creation. They only check that the supplied mint account is owned by the token program supplied to the terminal instruction. Consequently, a recreated mint can change the transfer behavior after the trade was accepted. For example, the mint authority can mint directly into the escrow and a later `SettleDvp`, `CancelDvp`, `RejectDvp` or `ReclaimDvp` transfer can credit less than the requested amount because Token-2022 withholds a transfer fee.

This requires control over a closeable zero-supply mint and the ability to recreate that mint account. It is still a concrete break in the one-time extension screening model because the recorded DvP state commits only to the mint address, not to the mint's accepted extension set.

Recommendation: Reject Token-2022 mints with `MintCloseAuthority` at `CreateDvp`,or record the accepted token program and extension-set commitment in `SwapDvp` and re-check them before terminal transfers. If closeable mints must remain supported, require clients to treat the close authority and mint account key holder as trusted parties for the entire lifetime of the DvP.

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Mint authorities can freeze or claw back escrows

Severity: Informational

Context: token_utils.rs#L96-L111

Description: `CreateDvp` does not reject mints with a freeze authority, including legacy SPL Token mints. It also allows Token-2022 mints with authority-controlled behavior such as `PermanentDelegate`, `Pausable` and `DefaultAccountState`. These features are intrinsic to some regulated assets, but they give the mint authority power over tokens after a DvP is created and funded.

For example, a permanent delegate can move escrowed tokens without the DvP PDA. A freeze authority can freeze the escrow ATA or set future ATAs to start frozen and a pause authority can pause transfers. These actions can make settlement fail or delay refunds until the mint authority cooperates. This is not an authorization bug in the DvP program, but it is a material integration risk for users who assume the escrow alone controls token movement.

Recommendation: Expose these mint-authority assumptions clearly to clients before funding. For stricter deployments, reject or require explicit allowlisting for legacy or Token-2022 mints with active freeze authority and for Token-2022 mints with `PermanentDelegate`, `Pausable` or `DefaultAccountState`. Settlement UIs should show these authorities as trusted parties, not as ordinary SPL-style mints.

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 Closed-account lamports go to terminal signers

Severity: Informational

Context: reject_dvp.rs#L169-L195

Description: `CreateDvp` lets any payer fund the `SwapDvp` PDA and both escrow ATAs, but the program does not record that payer as the rent recipient. When `RejectDvp` closes the trade, it sends both escrow ATA rents and the `SwapDvp` PDA lamports to the rejecting party.

Consequently, a counterparty can reject an unfunded or partially funded DvP and receive the account rent that a maker, relayer or other payer supplied during creation. The same close-recipient rule applies to any direct SOL lamport top-up sent to an escrow token account: SPL Token `CloseAccount` drains all lamports from the token account to the close destination, not to the original lamport sender or configured leg party.

This is not token loss and may be an intentional incentive choice, but it is a real value transfer that is not surfaced in the README's lifecycle or instruction table. `SettleDvp` and `CancelDvp` have the same payer/recipient split, except their close recipient is the settlement authority.

Recommendation: Document the account-lamport recipient model clearly for every terminal instruction, including rent and any direct lamport top-ups to escrow token accounts. If the create payer should recover rent, store a `rent_refund_recipient` in `SwapDvp` and send closed-account lamports there. If the current design is intentional, clients should show that the create payer or any direct SOL sender, is subsidizing the settlement authority or rejecting party.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. The README now documents that closed-account rent and any direct lamport top-ups go to the closing party: the settlement authority on `SettleDvp` / `CancelDvp` and the rejecting signer on `RejectDvp`. This resolves the undocumented value-transfer issue while preserving the intentional close-recipient model.

3.3.4 Confidential transfer fee denial is undocumented

Severity: Informational

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

Description: The code rejects Token-2022 mints with `ConfidentialTransferFeeConfig`, but the public deny-list documentation does not name that extension. The README says `CreateDvp` rejects `TransferFee`, `InterestBearing`, `ScaledUiAmount` and `ConfidentialTransfer`. The error table repeats the same list. The processor checks both `ConfidentialTransferMint` and `ConfidentialTransferFeeConfig`.

The integration test header also claims negative coverage for every blocked extension, but it only lists and tests `ConfidentialTransferMint` for the confidential-transfer family. Therefore a mint with `ConfidentialTransferFeeConfig` can be rejected by the program without being covered by the documented extension policy or by a dedicated regression test.

Recommendation: Update the README and generated documentation to list `ConfidentialTransferFeeConfig` explicitly in the blocked Token-2022 extension set.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. The README Token-2022 note and README error table now list `ConfidentialTransferFeeConfig` in the blocked Token-2022 extension set and the Token-2022 test header explains that the confidential-transfer test covers the fee-config path because that extension cannot exist independently of `ConfidentialTransferMint`.

3.3.5 `TransferHook` account cap can brick terminal instructions

Severity: Informational

Context: token_utils.rs#L140-L154

Description: `transfer_checked_cpi` caps transfer-hook remaining accounts at 32 per leg because the CPI account arrays are stack allocated. Token-2022 transfer-hook authorities can update an `ExtraAccountMetaList` after a DvP has been created and funded.

If a hook-bearing mint starts requiring more accounts than this cap, every terminal instruction that needs to transfer that leg fails before the token CPI can run. The result is a mint-authority-controlled denial of service: settlement, cancellation, rejection and reclaim remain blocked until the hook authority reduces the required account list. The source comments identify this as residual risk, but it is still a material integration constraint for users and integrators.

Recommendation: Treat mutable transfer-hook mints as explicitly trusted integrations. For stronger on-chain protection, reject hook-bearing mints unless the hook authority is unset or allowlisted by the parties. At minimum, surface this constraint in client-side risk checks before users fund an escrow.

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.3.6 Transfer hooks can receive caller signer privileges

Severity: Informational

Context: token_utils.rs#L191-L202

Description: `transfer_checked_cpi` builds the `TransferChecked` CPI account list by taking every trailing transfer-hook account and converting it with `InstructionAccount::from(acc)`. That conversion keeps the account's existing signer and writable flags.

That means the DvP program does not strip privileges from hook extras before handing them to Token-2022. If a client includes the user's signer account, the settlement authority or another transaction signer as a hook extra, the hook program receives that account as a signer during the hook call. If the extra is writable, the hook receives it as writable too.

This is not automatically a program bug. Transfer hooks are allowed to ask for extra accounts and trusted hooks may need signer or writable accounts to do their job. The problem is that a wallet or integration can hide this detail from the user. A user may think they are only signing a DvP settlement, cancellation, rejection or reclaim, while the transaction also gives a hook program access to one of their signer accounts.

This does not appear to expose the DvP PDA authority itself. Even if the hook asks for the transfer authority again, Token-2022 treats that fixed authority as read-only and not a signer before the hook runs. The real concern is with other extra accounts supplied by the caller. Those accounts can still reach the hook as signers or writable accounts.

Recommendation: Reject signer-bearing transfer-hook extras unless the integration explicitly needs them and has allowlisted the hook. At minimum, client tooling should warn before it includes any transaction signer or unrelated writable account in the hook extras.

Treat hook-bearing mints as active integrations with their own trust assumptions. They should not be presented to users as ordinary passive SPL Token mints.

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.3.7 README omits Token-2022 funding requirements

Severity: Informational

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

Description: The README says each side funds a DvP by issuing a plain SPL transfer to the escrow ATA. That is incomplete for accepted Token-2022 mints whose transfers require mint-aware or extension-aware account inputs.

For `Pausable` mints, Token-2022 creates `PausableAccount` token accounts. During an unchecked transfer, Token-2022 rejects those accounts because the mint account was not supplied and the token program cannot check whether the mint is paused. The DvP program itself uses `TransferChecked` for terminal transfers, but the documented funding step tells integrators to use a plain transfer.

For `TransferHook` mints, funding the escrow is also a token transfer, so Token-2022 invokes the hook during funding. The funding transaction must resolve and supply the hook program, validation PDA and any `ExtraAccountMetaList` accounts. A plain unchecked transfer can also fail for hook-bearing accounts because Token-2022 needs the mint account to resolve the hook.

Consequently, a client can follow the README and fail to fund Token-2022 mints that `CreateDvp` accepts and that the terminal instructions can support when the right accounts are supplied. The integration tests already reflect this for hook-bearing mints by funding them with `TransferChecked` plus hook extras, but the public funding documentation does not state the same requirement.

Recommendation: Update the funding section to state that Token-2022 mints with extension-aware transfer behavior must be funded with `TransferChecked` or an equivalent checked token instruction. The `Pausable` section should explicitly say that clients must include the mint account during funding so Token-2022 can enforce the current pause state. The `TransferHook` section should say that funding requires the same hook account-resolution logic as terminal transfers, including the hook program, validation PDA and resolved extra accounts.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. The README funding notes now instruct clients to use `TransferChecked` for Token-2022 mints with extension-aware transfer behavior, call out `Pausable` mint-account requirements and state that `TransferHook` funding needs the hook program, validation PDA and resolved `ExtraAccountMetaList` accounts.

3.3.8 SBF build leaves deploy keypairs that do not match declared program IDs

Severity: Informational

Context: Makefile#L18-L28

Description: `make build` regenerates the IDL and clients, then runs `cargo-build-sbf` for both the DvP program and the transfer-hook fixture, but it never verifies the generated `target/deploy/*-keypair.json` files against the program IDs compiled into the binaries and exposed to clients.

After a successful `make build`, the DvP deploy keypair resolved to `FvyU1M9YpbvZz1Joj57zyfCgt8wGY1F7VLhwXi2iBsnm`, while `program/src/lib.rs`, the README, the IDL and generated clients all use `DzG1qJupt6Khm8s8jB3p93NkhPoiAg2M7vkEhkS15CtC`. The transfer-hook fixture had the same mismatch: the deploy keypair resolved to `G6sc2dJebJN747T3qNxEBqzf4ezb8WdoZd4KXv5Ym9fw`, while the fixture declares `HookqJupt6Khm8s8jB3p93NkhPoiAg2M7vkEhkS15CtC`.

The `LiteSVM` tests do not catch this because they load the compiled `.so` bytes at fixed in-memory program IDs rather than deploying through the generated keypair files. A deployment workflow that relies on `target/deploy/*-keypair.json` or that omits an explicit `--program-id` keypair matching `declare_id!`, can publish a binary at an address that the README, IDL and generated clients do not use.

Recommendation: Add a build or deploy verification step that runs `solana-keygen pubkey target/deploy/<program>-keypair.json` and compares the result with each program's `declare_id!` and generated client ID. Fail CI and deployment if they differ. For real deployments, require an explicit `solana program deploy --program-id <expected-keypair.json>` path whose public key matches the documented program ID.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. The Makefile now includes `verify-program-id`, which compares the deploy keypair public key with the program ID from the generated IDL, and the README deployment flow requires running that guard before `solana program deploy --program-id`. This gives deployments an explicit mismatch check instead of relying on the random keypair emitted by `cargo-build-sbf`.

3.3.9 IDL does not describe `TransferHook` extra accounts

Severity: Informational

Context: dvp_swap_program.json#L377

Description: The Codama IDL is the JSON interface file for the program. Client generators, indexers, transaction builders and reviewers use it to learn which instructions exist, which accounts each instruction expects and which arguments must be encoded.

Here, the IDL is missing part of the account contract. It lists the normal fixed accounts for `reclaimDvp`, `settleDvp`, `cancelDvp` and `rejectDvp`, but it does not say that these instructions may also need extra accounts at the end of the account list.

Those extra accounts are needed for Token-2022 mints that use `TransferHook`. A `TransferHook` is a Token-2022 extension that lets the mint call another program during a token transfer. To run that hook, Token-2022 may need extra accounts such as the hook program, the hook validation account and any accounts listed by the hook's `ExtraAccountMetaList`.

The on-chain code expects those accounts even though the IDL does not document them. `ReclaimDvp` treats every account after its six fixed accounts as hook extras for the reclaim transfer. `SettleDvp`, `CancelDvp` and `RejectDvp` also accept trailing hook accounts. For those two-leg instructions, `legAExtrasCount` tells the program how many trailing accounts belong to leg A, the rest belong to leg B.

As a result, a client built only from `idl/dvp_swap_program.json` can produce incomplete transactions for hook-bearing mints. A transaction reviewer or indexer can also miss that the instruction may legally include these extra accounts. The generated TypeScript builder and parser issues are symptoms of the same root problem: the checked-in IDL does not fully describe the instruction interface.

Recommendation: Add Codama metadata, generation visitors or instruction documentation that describes the variable trailing accounts.

`ReclaimDvp` should expose one ordered list of hook extra accounts. `SettleDvp`, `CancelDvp` and `RejectDvp` should expose separate hook-extra lists for leg A and leg B, then derive `legAExtrasCount` from those lists.

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.

3.3.10 Unbounded `expiry_timestamp` permits indefinite escrow lifetime and permanent rent lock-up

Severity: Informational

Context: create_dvp.rs#L263

Description: `CreateDvp` validates that expiry_timestamp > now (create_dvp.rs:263-266) but imposes no upper bound on how far in the future it may be set. Because `expiry_timestamp` is an `i64`, a caller can specify any value up to `i64::MAX`, and `validate_args` will accept it. The same lack of bound applies to `earliest_settlement_timestamp` indirectly and it only has to satisfy `earliest <= expiry`, so it inherits the same ceiling.

Recommendation: Add an upper bound on `expiry_timestamp` in `validate_args`, expressed as a maximum duration from now rather than an absolute timestamp so the limit doesn't drift over time.

Pick the horizon based on the product's longest legitimate settlement window. For institutional `T+N` DvP that's typically days to weeks, so a one-year cap is generous and still defeats the `i64::MAX` griefing case.

Solana Foundation: Fixed in commit c149cdf1.

Cantina Managed: Fix verified. `validate_args` now rejects `expiry_timestamp` values more than one year after creation with `ExpiryTooFarInFuture`, and `earliest_settlement_timestamp` remains bounded by `expiry_timestamp`.

3.3.11 Settlement window may be configured too narrow to be operationally usable

Severity: Informational

Context: create_dvp.rs#L257

Description: `validate_args` enforces `earliest_settlement_timestamp <= expiry_timestamp` but does not require a minimum gap between the two. As a result, callers can create a DvP with `earliest == expiry` (a single-slot settle window) or `earliest == expiry - 1` (a ~1-second window).

Settle then requires `now >= earliest && now <= expiry`, and given that `Clock::unix_timestamp` advances in integer seconds and Solana slot timing plus propagation jitter routinely costs more than that, the window can be missed even by a correctly-submitted transaction.

Funds remain recoverable via `Reject`, so this is not a fund-loss issue but a foot-gun where a misconfigured client silently produces a DvP that is mathematically valid but practically unsettleable.

Recommendation: Add a minimum-window check in `validate_args`, applied only when `earliest_settlement_timestamp` is set:

```
const MIN_SETTLE_WINDOW_SECS: i64 = 600;

if let Some(earliest) = args.earliest_settlement_timestamp {
    require!(
      args.expiry_timestamp.saturating_sub(earliest) >= MIN_SETTLE_WINDOW_SECS,
          DvpSwapProgramError::SettleWindowTooNarrow
      );
}
```

Solana Foundation: Acknowledged.

Cantina Managed: Acknowledged.