



TECHNICAL SECURITY ASSESSMENT

Corvid AMM

Smart Contract Audit · DEX

REPORT ID QX-TSA-2026-003

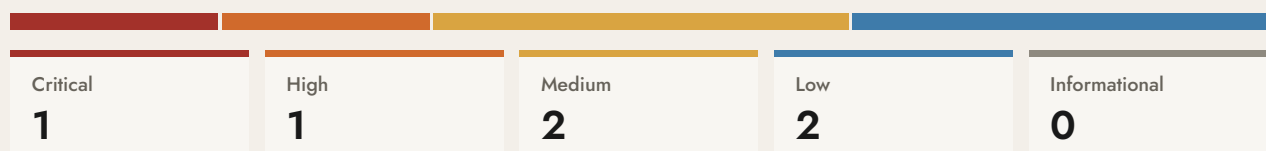
QUANTIX RESEARCH ASSESSMENT

Pool initialisation, swaps, liquidity deposits and withdrawals, and fee collection. 2,310 lines of Rust.

Document status	Published
Published	2 June 2026
Review period	Ecosystems
11 May 2026 – 20 May 2026	Solana
Languages	Review effort
Rust (Anchor)	Two reviewers, eight working days
Reviewed commit	Remediation commit
b1a8ddef6dd1966c179277f2f929dbf74114494e	5343f231c925b19fbf5a2f800d728f95337ab06e

FINDINGS BY SEVERITY

6 findings · 100% remediated



This document is a Quantix research assessment of Corvid AMM, a protocol design from Quantix Protocol Research. It is not a client engagement, it does not describe a deployed system, and it is not a certification.

DOCUMENT CONTROL

Document ID	QX-TSA-2026-003
Title	Technical Security Assessment: Corvid AMM
Version	1.1
Status	Published
Assessment type	Smart Contract Audit
Review period	11 May 2026 – 20 May 2026
Published	2 June 2026
Reviewed commit	b1a8ddf6dd1966c179277f2f929dbf74114494e

CONTENTS

1	Executive summary	3
2	Scope	4
3	Protocol architecture	4
4	Threat model	5
5	Methodology and severity classification	6
6	Findings summary	7
7	Detailed findings	8
	CA-01 Withdraw does not verify that the LP mint belongs to the pool	8
	CA-02 Fee authority is compared but not required to sign	9
	CA-03 Constant-product calculation can overflow in release builds	10
	CA-04 Swap fee rounds down to zero for small trades	11
	CA-05 Deposits have no minimum LP output	12
	CA-06 First depositor sets the initial price without bounds	12
8	Severity and risk analysis	14
9	Remediation status	14
10	Formal verification and invariants	15
11	Recommendations	16
12	Limitations	16
A	Technical appendix	17

1

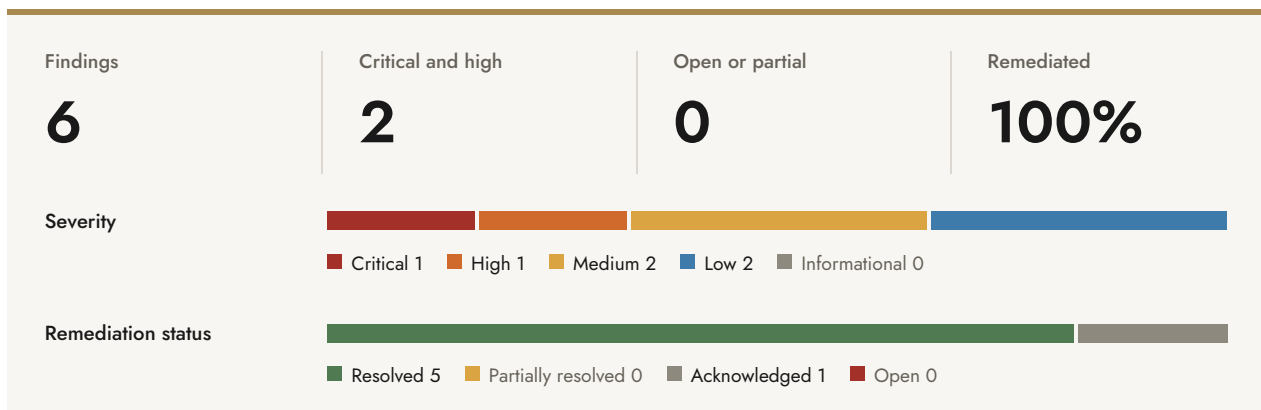
Executive summary

Corvid AMM is a constant-product market maker on Solana built with Anchor. On Solana every account an instruction touches is supplied by the caller, so the review concentrated on whether each account is constrained to the one the program expects.

Six findings were reported. The critical issue allowed an attacker to burn tokens from a mint they control and withdraw the pool's reserves, because the withdrawal instruction did not check that the LP mint belonged to the pool. A high-severity issue let anyone withdraw protocol fees because the fee authority was compared but never required to sign.

All critical, high and medium findings were fixed and verified, and negative tests were added for every account constraint. One low finding about pool initialisation was acknowledged with a front-end mitigation.

AT A GLANCE



SEVERITY SUMMARY

Severity	Definition	Count	Open
■ Critical	Direct loss of funds or permanent loss of control is achievable with little or no precondition.	1	0
■ High	Material loss or protocol disruption is achievable under realistic conditions.	1	0
■ Medium	Limited loss, degraded functionality, or impact that depends on specific conditions.	2	0
■ Low	Minor impact, defence-in-depth gaps, or issues with unlikely preconditions.	2	0
■ Informational	Code quality, documentation or best-practice observations without direct security impact.	0	0

2

Scope

Pool initialisation, swaps, liquidity deposits and withdrawals, and fee collection. 2,310 lines of Rust.

Repository	github.com/example-org/corvid-amm
Reviewed commit	b1a8ddef6dd1966c179277f2f929dbf74114494e
Remediation commit	5343f231c925b19fbf5a2f800d728f95337ab06e
Review period	11 May 2026 – 20 May 2026
Review effort	Two reviewers, eight working days

Path	Description	nSLOC
programs/corvid-amm/src/instructions/initialize_pool.rs	Pool creation, PDA derivation and vault setup	286
programs/corvid-amm/src/instructions/swap.rs	Constant-product swap with fees and slippage checks	412
programs/corvid-amm/src/instructions/deposit.rs	Liquidity deposits and LP token minting	344
programs/corvid-amm/src/instructions/withdraw.rs	LP token burning and proportional withdrawal	318
programs/corvid-amm/src/instructions/collect_fees.rs	Protocol fee withdrawal	142
programs/corvid-amm/src/state.rs	Pool account layout and invariant helpers	230
Total		1,732

Out of scope

- TypeScript SDK and front end
- Upgrade-authority management of the deployed program
- The SPL Token program and Anchor framework (assumed to behave as documented)

3

Protocol architecture

Each pool is a program-derived account (PDA) seeded by the two token mints. The pool PDA is the authority over two SPL token vaults that hold the reserves, and over the LP mint.

Swaps transfer the input token into one vault and the computed output from the other, preserving $\text{reserve}_a \times \text{reserve}_b$ after fees. Deposits mint LP tokens in proportion to the contribution; withdrawals burn LP tokens and return a proportional share of both vaults.

A fee authority stored in the pool account can withdraw accumulated protocol fees, tracked separately from LP

reserves.

Component	Role	Trust level
corvid_amm program	Instruction handlers and invariant checks	Trusted
Pool PDA	Pool state; authority over vaults and LP mint	Trusted
Token vaults	SPL token accounts holding reserves	Trusted
LP mint	Mint for liquidity provider tokens	Trusted
Caller-supplied accounts	Every account passed to an instruction	Untrusted
Fee authority	Withdraws protocol fees	Trusted

Trust assumptions

- The program upgrade authority is held by the protocol team and not compromised.
- Pooled tokens use the standard SPL Token program; Token-2022 extensions such as transfer fees and hooks are not supported.
- The fee authority key is held securely.

Threat model

Assets

- Pool reserves in both vaults
- Accrued protocol fees
- LP token supply integrity

Actors

Actor	Capability	Trust
Trader	Calls swap with any accounts	Untrusted
Liquidity provider	Deposits and withdraws with any accounts	Untrusted
Fee authority	Withdraws protocol fees	Trusted

Threats

Threat	Actor and vector	Mitigation	Findings
Reserve drain through a substituted LP mint	Liquidity provider: Passing an attacker-controlled mint to withdraw	Address and has_one constraints	CA-01
Unauthorised fee withdrawal	Trader: Passing the authority key without its signature	Signer constraint	CA-02
Invariant bypass through overflow	Trader: Large reserves with high-decimal tokens	u128 checked arithmetic	CA-03

Threat	Actor and vector	Mitigation	Findings
Fee avoidance	Trader: Splitting trades below the rounding threshold	Multiply before divide, round up	CA-04

5

Methodology and severity classification

1. Manual review of every instruction and its account constraints
2. Account-substitution analysis: for each account, what happens if the caller supplies a different one
3. Property-based tests of the constant-product invariant across swap and liquidity sequences
4. Arithmetic review of all integer operations under release-build overflow semantics
5. Remediation review against the fix commit, including negative tests for each constraint

Severity classification

Each finding is rated on the impact it would have if exploited and the likelihood that its preconditions occur. The rating reflects the code at the reviewed commit.

Severity	Meaning
 Critical	Direct loss of funds or permanent loss of control is achievable with little or no precondition.
 High	Material loss or protocol disruption is achievable under realistic conditions.
 Medium	Limited loss, degraded functionality, or impact that depends on specific conditions.
 Low	Minor impact, defence-in-depth gaps, or issues with unlikely preconditions.
 Informational	Code quality, documentation or best-practice observations without direct security impact.

Remediation status

Status	Meaning
Resolved	Fixed and verified in the remediation review.
Partially resolved	Mitigated; the residual risk is documented.
Acknowledged	Accepted by the protocol team with a stated rationale.
Open	Not yet addressed at the reviewed remediation commit.

6

Findings summary

ID	Title	Severity	Category	Status
CA-01	Withdraw does not verify that the LP mint belongs to the pool	 Critical	Account validation	Resolved
CA-02	Fee authority is compared but not required to sign	 High	Access control	Resolved
CA-03	Constant-product calculation can overflow in release builds	 Medium	Precision and rounding	Resolved
CA-04	Swap fee rounds down to zero for small trades	 Medium	Precision and rounding	Resolved
CA-05	Deposits have no minimum LP output	 Low	Business logic	Resolved
CA-06	First depositor sets the initial price without bounds	 Low	Business logic	Acknowledged

7

Detailed findings

CA-01 ■ Critical Resolved

Withdraw does not verify that the LP mint belongs to the pool

programs/corvid-amm/src/instructions/withdraw.rs:L18-L52

Category	CWE	Likelihood	Impact
Account validation	CWE-345	High	High

DESCRIPTION

The `Withdraw` accounts struct declares `lp_mint: Account<'info, Mint>` without an `address = pool.lp_mint` or `has_one = lp_mint` constraint. Anchor therefore verifies only that the account is a valid mint owned by the SPL Token program, not that it is this pool's mint.

The share of reserves returned is computed as `lp_amount / lp_mint.supply` applied to both vaults, and the burn is performed against the supplied mint. An attacker can create a new mint, mint a single token to themselves, and pass that mint and token account to `withdraw`.

The program burns the attacker's one token from the fake mint, computes a share of $1/1 = 100\%$, and transfers the entire contents of both vaults.

PRECONDITIONS

- The attacker can create an SPL mint (anyone can)
- The pool holds non-zero reserves

EXPLOIT SCENARIO

1. Attacker creates mint M with themselves as mint authority and mints 1 token.
2. Attacker calls `withdraw` on the target pool, passing M as `lp_mint` and their M token account as the LP source.
3. The program burns 1 token of M and computes `share = 1 / supply(M) = 100%`.
4. Both vaults are transferred to the attacker's token accounts in one transaction.

IMPACT

Complete loss of both reserves in every pool, in a single transaction per pool, with no capital required.

RECOMMENDATION

Constrain `lp_mint` to `pool.lp_mint` (for example `#[account(mut, address = pool.lp_mint)]` or `has_one = lp_mint` on the pool), and add a negative test that passes a foreign mint and expects failure.

FIX

programs/corvid-amm/src/instructions/withdraw.rs	DIFF
1	<code>#[derive(Accounts)]</code>
2	<code>pub struct Withdraw<'info> {</code>
3	<code>- #[account(mut)]</code>
4	<code>+ #[account(mut, has_one = lp_mint, has_one = vault_a, has_one = vault_b)]</code>
5	<code>pub pool: Account<'info, Pool>,</code>
6	<code>- #[account(mut)]</code>
7	<code>+ #[account(mut, address = pool.lp_mint)]</code>
8	<code>pub lp_mint: Account<'info, Mint>,</code>

RESOLUTION

Address constraint added, plus `has_one` constraints for the mint and both vaults. Negative tests pass a foreign mint and foreign vaults and expect `ConstraintHasOne` errors.

REFERENCES

- Anchor account constraints
<https://www.anchor-lang.com/docs/references/account-constraints>
- Solana account model
<https://solana.com/docs/core/accounts>
- CWE-345: Insufficient Verification of Data Authenticity
<https://cwe.mitre.org/data/definitions/345.html>

CA-02 ■ High Resolved

Fee authority is compared but not required to sign

programs/corvid-amm/src/instructions/collect_fees.rs:L11-L29

Category	CWE	Likelihood	Impact
Access control	CWE-862	High	Medium

DESCRIPTION

`collect_fees` takes `authority: UncheckedAccount<'info>` and checks `authority.key() == pool.fee_authority`. The comparison proves only that the caller knows the authority's public key, which is public. The account is never required to sign the transaction.

On Solana, authorisation comes from the signer flag on the account meta. Without `Signer<'info>` (or an explicit `is_signer` check), any caller can satisfy the comparison.

PRECONDITIONS

- Accumulated protocol fees greater than zero

EXPLOIT SCENARIO

- Attacker reads `pool.fee_authority` from the pool account.
- Attacker calls `collect_fees` passing that public key as authority (unsigned) and their own token account as destination.
- Fees are transferred to the attacker.

IMPACT

Loss of all accumulated protocol fees across all pools. LP reserves are not affected because fees are tracked separately.

RECOMMENDATION

Declare the account as `Signer<'info>` and keep the key comparison through `has_one = fee_authority` on the pool.

FIX

programs/corvid-amm/src/instructions/collect_fees.rs	DIFF
1 <code>#[derive(Accounts)]</code>	
2 <code>pub struct CollectFees<'info> {</code>	
3 <code>- pub pool: Account<'info, Pool>,</code>	
4 <code>- /// CHECK: compared against pool.fee_authority</code>	
5 <code>- pub authority: UncheckedAccount<'info>,</code>	
6 <code>+ #[account(mut, has_one = fee_authority)]</code>	
7 <code>+ pub pool: Account<'info, Pool>,</code>	

```
8 + pub fee_authority: Signer<'info>,
```

RESOLUTION

Account changed to Signer with a has_one constraint; negative test with an unsigned authority added.

REFERENCES

- Anchor account constraints: signer and has_one
<https://www.anchor-lang.com/docs/references/account-constraints>
- CWE-862: Missing Authorization
<https://cwe.mitre.org/data/definitions/862.html>

CA-03 ■ Medium Resolved

Constant-product calculation can overflow in release builds

programs/corvid-amm/src/state.rs:L88-L104

Category	CWE	Likelihood	Impact
Precision and rounding	CWE-190	Low	High

DESCRIPTION

The invariant `reserve_a * reserve_b` is computed in `u64`. Rust release builds wrap on integer overflow unless `overflow-checks = true` is set in the release profile, and the workspace did not set it.

Two vaults of a 9-decimal token holding about 4.3 million whole tokens each already exceed `u64::MAX` when multiplied. At that point the computed product wraps to a small number and the invariant check after a swap compares against a meaningless value.

PRECONDITIONS

- Reserves large enough that the product exceeds 2^{64}
- Release build without overflow checks

EXPLOIT SCENARIO

- A pool grows past the overflow threshold.
- A swap's post-trade product wraps and passes the invariant check at an incorrect price.
- The trader receives more output than the curve allows.

IMPACT

Swaps can succeed at incorrect prices once reserves are large, extracting value from liquidity providers.

RECOMMENDATION

Compute in `u128` with `checked_mul`, return an explicit error on overflow, and enable `overflow-checks = true` in the release profile as defence in depth.

FIX

```
programs/corvid-amm/src/state.rs
```

DIFF

```
1 - let k = self.reserve_a * self.reserve_b;  
2 + let k = (self.reserve_a as u128)  
3 +   .checked_mul(self.reserve_b as u128)  
4 +   .ok_or(AmmError::MathOverflow)?;
```

RESOLUTION

Moved to u128 checked arithmetic throughout; overflow-checks enabled in Cargo.toml.

REFERENCES

- Cargo profiles: overflow-checks
<https://doc.rust-lang.org/cargo/reference/profiles.html#overflow-checks>
- CWE-190: Integer Overflow or Wraparound
<https://cwe.mitre.org/data/definitions/190.html>

CA-04 ■ Medium Resolved

Swap fee rounds down to zero for small trades

programs/corvid-amm/src/instructions/swap.rs:L71

Category	CWE	Likelihood	Impact
Precision and rounding	CWE-682	High	Low

DESCRIPTION

The fee is computed as `amount_in / 10_000 * fee_bps`. Division before multiplication truncates the intermediate result, so any trade below 10,000 base units pays no fee at all, and larger trades are under-charged by up to `fee_bps - 1` units.

On Solana, transaction fees are low enough that splitting a large trade into many small ones can be cheaper than paying the swap fee.

PRECONDITIONS

- A non-zero `fee_bps` configuration

EXPLOIT SCENARIO

1. Trader splits a large swap into many trades just below 10,000 base units in one transaction.
2. Each trade pays zero fee.

IMPACT

Liquidity providers lose fee income; value transfers to traders who split orders.

RECOMMENDATION

Multiply before dividing and round the fee up: `(amount_in * fee_bps + 9_999) / 10_000` computed in u128.

FIX

programs/corvid-amm/src/instructions/swap.rs	DIFF
<pre>1 - let fee = amount_in / 10_000 * fee_bps as u64; 2 + let fee = ((amount_in as u128) * (fee_bps as u128) + 9_999) / 10_000; 3 + let fee = u64::try_from(fee).map_err(_ AmmError::MathOverflow)?;</pre>	

RESOLUTION

Formula corrected with upward rounding; PB-03 holds.

REFERENCES

- CWE-682: Incorrect Calculation
<https://cwe.mitre.org/data/definitions/682.html>

CA-05 ■ Low Resolved

Deposits have no minimum LP output

programs/corvid-amm/src/instructions/deposit.rs:L40

Category	CWE	Likelihood	Impact
Business logic	—	Medium	Low

DESCRIPTION

Swaps enforce `minimum_amount_out`, but deposits accept any LP amount. A deposit that lands after a large price move (or after a sandwiching trade) mints fewer LP tokens than the user was quoted.

PRECONDITIONS

- Price moves between quote and execution

EXPLOIT SCENARIO

1. A searcher moves the pool price before a deposit lands.
2. The deposit mints fewer LP tokens; the searcher reverses the trade.

IMPACT

Depositors can receive materially fewer LP tokens than quoted.

RECOMMENDATION

Add a `min_lp_out` argument and revert when it is not met.

FIX

programs/corvid-amm/src/instructions/deposit.rs	DIFF
1 - pub fn deposit(ctx: Context<Deposit>, amount_a: u64, amount_b: u64) -> Result<> {	
2 + pub fn deposit(ctx: Context<Deposit>, amount_a: u64, amount_b: u64, min_lp_out: u64) -> Result<> {	
3 let lp_out = ctx.accounts.pool.lp_for(amount_a, amount_b)?;	
4 + require!(lp_out >= min_lp_out, AmmError::SlippageExceeded);	

RESOLUTION

Parameter added; SDK passes quoted values with a tolerance.

REFERENCES

- Solana programs: instruction design
<https://solana.com/docs/core/programs>

CA-06 ■ Low Acknowledged

First depositor sets the initial price without bounds

programs/corvid-amm/src/instructions/initialize_pool.rs:L33

Category	CWE	Likelihood	Impact
Business logic	—	Medium	Low

DESCRIPTION

Pool creation and the first deposit are separate instructions. Between them, any third party can make the first deposit at an arbitrary ratio, setting the pool's opening price.

PRECONDITIONS

- A pool is created without an atomic first deposit

EXPLOIT SCENARIO

1. A pool is initialised.
2. A third party deposits at a skewed ratio before the creator.
3. The first traders trade at a misleading price until arbitrage corrects it.

IMPACT

The pool opens at a misleading price; the cost falls on whoever trades first.

RECOMMENDATION

Combine initialisation with the first deposit, or restrict the first deposit to the pool creator.

RESOLUTION

Acknowledged. The front end creates and seeds pools atomically in one transaction; the program behaviour is documented for integrators.

REFERENCES

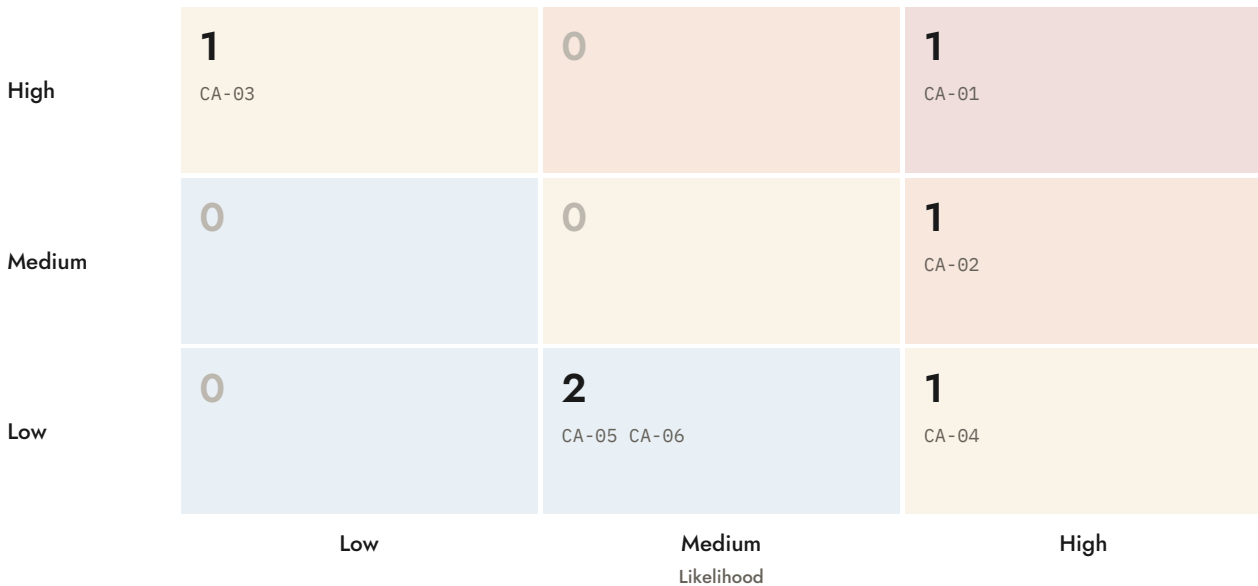
- Solana transactions: atomic instruction execution
<https://solana.com/docs/core/transactions>

8

Severity and risk analysis

The matrix places each finding by the likelihood of its preconditions and the impact of exploitation. Identifiers in each cell refer to the detailed findings in section 7.

Impact



Distribution by vulnerability class



Severity distribution



9

Remediation status

5 of 6 findings are resolved, 1 acknowledged, 0 partially resolved and 0 open, as reviewed at remediation commit 5343f231c925.

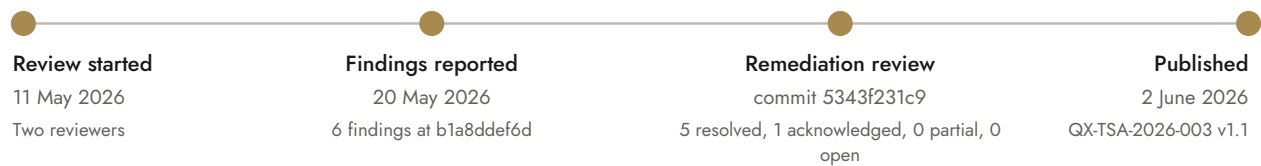
Severity	Resolved	Partially resolved	Acknowledged	Open	Total
Critical	1	0	0	0	1
High	1	0	0	0	1

Severity	Resolved	Partially resolved	Acknowledged	Open	Total
■ Medium	2	0	0	0	2
■ Low	1	0	1	0	2
■ Informational	0	0	0	0	0

Remediation progress

100%

Remediation history



Residual items

ID	Status	Notes
CA-06	Acknowledged	Acknowledged. The front end creates and seeds pools atomically in one transaction; the program behaviour is documented for integrators.

10

Formal verification and invariants

No formal verification was performed; Solana program verification tooling was not in scope. The constant-product and LP-share properties below were checked with property-based tests over random instruction sequences.

Tool Property-based testing (Rust)

ID	Property	Result	Notes
PB-01	reserve_a × reserve_b never decreases across a swap (after fees)	■ Inconclusive	Held across 1M random sequences after the CA-03 fix.
PB-02	Withdrawn amounts never exceed the caller's LP share of reserves	■ Inconclusive	Violated on the original commit via CA-01; held after the fix.
PB-03	Every swap pays a fee of at least one base unit when fee_bps > 0	■ Inconclusive	Violated on the original commit (CA-04); held after the fix.

11

Recommendations

Constrain every account

Adopt a rule that every account in every instruction has an explicit owner, address, seeds or has_one constraint, and keep a negative test for each.

Support Token-2022 deliberately

If Token-2022 mints are to be supported, review transfer fees, transfer hooks and confidential transfers before enabling them; until then reject them explicitly.

Protect the upgrade authority

The program is upgradeable. Hold the upgrade authority in a multisig and consider a timelocked upgrade process.

12

Limitations

- Only the program code at the listed commit was reviewed. The TypeScript SDK and front end were out of scope.
- The SPL Token program and Anchor framework were treated as behaving as documented.
- Upgrade-authority management for the deployed program was not assessed.

No assessment can show that a system is free of vulnerabilities. Conclusions apply to the code at the commits listed and the conditions stated in this document.

Appendix

Technical appendix

A.1 Tools

Tool	Purpose
Anchor test framework	Integration and negative constraint tests
Property-based testing	Invariant checks over random instruction sequences
Manual review	Account-substitution analysis for every instruction

A.2 Reviewed files (SHA-256)

File	SHA-256
programs/corvid-amm/src/instructions/withdraw.rs	7e3b94d8b5e77e27cdeb167637de152fcd6dfae823ea00e3a481aebbb1008aa4
programs/corvid-amm/src/instructions/swap.rs	9d1afef9355e0b767d90b8304fa083f1105fed921d941ec14371b5cbd91e07cd
programs/corvid-amm/src/instructions/deposit.rs	9ceb8b0c3ef7737279d2466be046aad07b3f629640e2a9597da13a7193d51130
programs/corvid-amm/src/instructions/collect_fees.rs	338704eceed1f358ec2c173a274e993ad059ad650a5997b4f695a749e6997b8a
programs/corvid-amm/src/state.rs	1e99070596a7d5325eb31e7682284ead71c6f296f472e3ac147925c11ba35926

A.3 Notes

- Program built with Rust 1.79 and Anchor 0.30.

A.4 Methodology and technical references

Standards, specifications and weakness classifications cited in the findings. Each entry links to the official source.

1. Anchor account constraints
<https://www.anchor-lang.com/docs/references/account-constraints>
2. Solana account model
<https://solana.com/docs/core/accounts>
3. CWE-345: Insufficient Verification of Data Authenticity
<https://cwe.mitre.org/data/definitions/345.html>
4. CWE-862: Missing Authorization
<https://cwe.mitre.org/data/definitions/862.html>
5. Cargo profiles: overflow-checks
<https://doc.rust-lang.org/cargo/reference/profiles.html#overflow-checks>
6. CWE-190: Integer Overflow or Wraparound
<https://cwe.mitre.org/data/definitions/190.html>
7. CWE-682: Incorrect Calculation
<https://cwe.mitre.org/data/definitions/682.html>
8. Solana programs: instruction design
<https://solana.com/docs/core/programs>

9. Solana transactions: atomic instruction execution
<https://solana.com/docs/core/transactions>

A.5 Document information

Document	Quantix Technical Security Assessment QX-TSA-2026-003, version 1.1
Subject	Corvid AMM, Quantix Protocol Research
Status	Published 2 June 2026
Disclosure	This document is a Quantix research assessment of Corvid AMM, a protocol design from Quantix Protocol Research. It is not a client engagement, it does not describe a deployed system, and it is not a certification.