



TECHNICAL SECURITY ASSESSMENT

Veyra Finance

Smart Contract Audit · Lending

REPORT ID QX-TSA-2026-007

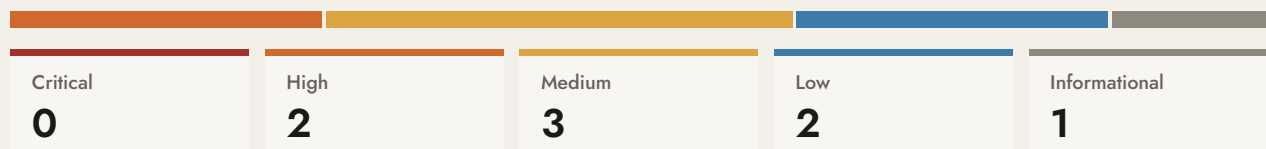
QUANTIX RESEARCH ASSESSMENT

Lending pool, vTokens with permit, price oracle, liquidator and treasury. 1,927 nSLOC of Solidity on Base.

Document status	Published
Published	24 March 2026
Review period	Ecosystems
9 February 2026 — 27 February 2026	Base
Languages	Review effort
Solidity	Three reviewers over fifteen working days, followed by a five-day remediation review
Reviewed commit	Remediation commit
7c1e94b0a3d25f86e0b4c97a12d8f63e5b0a9c41	d04a8f2c6b19e73a5f0c8d2e41b67a9c3e5f1082

FINDINGS BY SEVERITY

8 findings · 94% remediated



This document is a Quantix research assessment of Veyra Finance, 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-2026-007
Title	Technical Security Assessment: Veyra Finance
Version	1.1
Status	Published
Assessment type	Smart Contract Audit
Review period	9 February 2026 – 27 February 2026
Published	24 March 2026
Reviewed commit	7c1e94b0a3d25f86e0b4c97a12d8f63e5b0a9c41

CONTENTS

1	Executive summary	3
2	Scope	4
3	Protocol architecture	5
4	Threat model	5
5	Methodology and severity classification	6
6	Findings summary	7
7	Detailed findings	8
	VF-01 Read-only reentrancy lets flash-loan callbacks liquidate vToken collateral at a depressed exchange rate	8
	VF-02 Oracle falls back to a manipulable AMM spot price whenever the primary feed exceeds a one-hour s...	9
	VF-03 vToken permit domain separator uses a hard-coded chain ID and omits the verifying contract	11
	VF-04 Front-running the permit inside depositWithPermit blocks the deposit	12
	VF-05 No grace period after the Base sequencer recovers before prices are used for liquidations	13
	VF-06 Reserve-manager role remains with the deployment key	14
	VF-07 Share conversion rounds in the depositor's favour on mint and redeem	15
	VF-08 Liquidation events are emitted before state changes and lack indexed fields	16
8	Severity and risk analysis	17
9	Remediation status	18
10	Formal verification and invariants	18
11	Recommendations	19
12	Limitations	19
A	Technical appendix	20

1

Executive summary

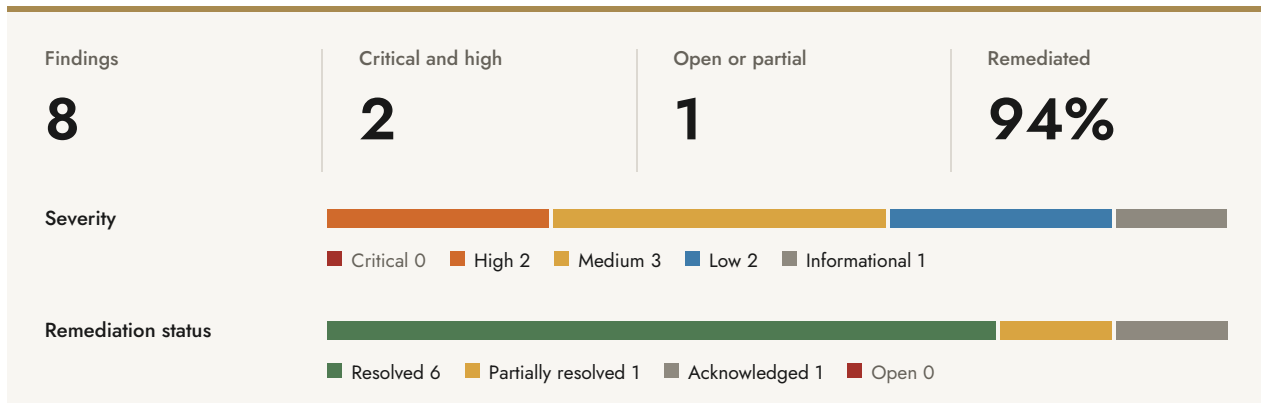
Veyra Finance is a pooled money market on Base. Depositors supply assets to shared pools and receive interest-bearing vTokens; borrowers draw against collateral priced by an aggregated oracle that checks the Base sequencer-uptime feed. The pool also offers single-transaction flash loans. Quantix reviewed the pool, token, oracle, liquidation and treasury contracts at the commit listed in this report.

The review identified two high-severity issues. The first is a read-only reentrancy: while a flash loan is outstanding the vToken exchange rate is temporarily depressed, and a separate liquidation contract that values vToken collateral could be invoked from inside the flash-loan callback to liquidate healthy accounts at a discount. The second is an oracle fallback that switched to an AMM spot price whenever the primary feed was older than a fixed one-hour threshold, which for slow-moving assets made a flash-loan manipulable price the effective source most of the time.

Three medium-severity findings concerned the vToken permit domain, the absence of a grace period after the sequencer recovers, and a reserve-management role left on a deployment key. All high and medium findings were fixed in the remediation commit and the fixes were verified, including new invariant tests that reproduce both high-severity issues against the original code.

Residual risk is concentrated in collateral listings: any new asset whose primary feed has a long heartbeat should be listed with an explicit staleness configuration, and markets that accept vTokens as collateral depend on the reentrancy check added for VF-01 remaining in place.

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.	0	0
■ High	Material loss or protocol disruption is achievable under realistic conditions.	2	0
■ Medium	Limited loss, degraded functionality, or impact that depends on specific conditions.	3	0
■ Low	Minor impact, defence-in-depth gaps, or issues with unlikely preconditions.	2	1

Severity	Definition	Count	Open
■ Informational	Code quality, documentation or best-practice observations without direct security impact.	1	0

2

Scope

Lending pool, vTokens with permit, price oracle, liquidator and treasury. 1,927 nSLOC of Solidity on Base.

Repository	github.com/example-org/veyra-protocol
Reviewed commit	7c1e94b0a3d25f86e0b4c97a12d8f63e5b0a9c41
Remediation commit	d04a8f2c6b19e73a5f0c8d2e41b67a9c3e5f1082
Review period	9 February 2026 – 27 February 2026
Review effort	Three reviewers over fifteen working days, followed by a five-day remediation review

Path	Description	nSLOC
src/LendingPool.sol	Deposits, borrows, repayments, interest accrual and flash loans	812
src/tokens/VToken.sol	Interest-bearing deposit receipts with EIP-2612 permit	286
src/oracle/PriceOracle.sol	Aggregated price feeds, staleness rules and sequencer-uptime check	214
src/liquidation/Liquidator.sol	Health checks and liquidation auctions	398
src/Treasury.sol	Reserve-factor accrual and reserve withdrawals	121
src/libraries/WadRayMath.sol	Fixed-point arithmetic helpers	96
Total		1,927

Out of scope

- Web front end, keeper and liquidation bots
- The Base sequencer, bridge and L1 contracts
- External price-feed providers, which were treated as behaving as documented
- Deployment of the governance multisig and its signer set

3

Protocol architecture

The `LendingPool` holds every market's cash, tracks borrows with a per-market borrow index, and mints and burns `VToken` receipts. Interest accrues lazily on every state-changing call through `accrueInterest(market)`.

Collateral value comes from `PriceOracle`, which reads a primary feed per asset, rejects prices older than a configured staleness window, and refuses all prices while the Base sequencer is reported down. The `Liquidator` is a separate contract that checks account health through the pool and runs liquidation auctions; it has its own reentrancy guard.

`vTokens` of one market may be used as collateral in another, valued at `exchangeRateStored()` × underlying price. This composability is the root of the read-only reentrancy described in VF-01.

Component	Role	Trust level
<code>LendingPool</code>	Market accounting, flash loans, interest accrual	Trusted
<code>VToken</code>	Deposit receipts with permit	Trusted
<code>PriceOracle</code>	Collateral and debt pricing	Trusted
<code>Liquidator</code>	Liquidations and auctions	Trusted
<code>Treasury</code>	Reserve accrual and withdrawals	Trusted
Primary price feeds	External price source	Semi-trusted
Flash-loan receivers, liquidators, depositors	Any external account	Untrusted

Trust assumptions

- Primary price feeds report honest prices within their documented heartbeat and deviation thresholds.
- The governance multisig and timelock act in the interest of users; parameter changes pass through a 48-hour timelock.
- The Base sequencer-uptime feed reflects sequencer status accurately.
- Listed assets are standard ERC-20 tokens without transfer fees or rebasing.

4

Threat model

Assets

- Pooled deposits in every market
- Collateral of borrowers
- Protocol reserves in the Treasury
- Integrity of `vToken` exchange rates used by integrators

Actors

Actor	Capability	Trust
Flash-loan attacker	Borrows large amounts within one transaction and calls any external function	Untrusted
Liquidator	Calls liquidation functions on any unhealthy account	Untrusted
Permit relayer	Observes and submits signed permits from the mempool	Untrusted
Governance multisig	Lists markets and changes risk parameters through the timelock	Trusted
Deployer key	Deploys contracts and initial roles	Semi-trusted

Threats

Threat	Actor and vector	Mitigation	Findings
Liquidate healthy accounts using a transiently wrong price	Flash-loan attacker: Re-enter a separate contract during the flash-loan callback	Revert views while the pool is entered; account for outstanding flash loans	VF-01
Borrow against inflated collateral	Flash-loan attacker: Skew an AMM pool used as a fallback price source	No spot-price fallback; per-feed staleness matching heartbeat	VF-02
Replay a permit signature on another deployment	Permit relayer: Domain separator without chain-specific values	EIP-712 domain with chain ID and verifying contract	VF-03
Block a user deposit	Permit relayer: Front-run the permit embedded in a deposit	Tolerate an already-consumed permit	VF-04
Liquidations at stale prices after sequencer recovery	Liquidator: Act before users can respond to price moves that happened during downtime	Grace period after the sequencer comes back up	VF-05
Withdraw protocol reserves	Deployer key: Use a role that was never revoked	Roles held only by the timelock	VF-06

5

Methodology and severity classification

1. Line-by-line manual review of every in-scope contract by at least two reviewers
2. Specification walk-through with the Veyra team covering interest accrual, liquidation incentives and oracle configuration
3. Static analysis with triage of all detector output
4. Stateful invariant fuzzing of pool accounting, including flash loans and cross-market vToken collateral
5. Economic analysis of oracle fallbacks and liquidation incentives under flash-loan capital
6. Bounded symbolic tests for flash-loan repayment and borrow health properties
7. Remediation review of the fix commit with regression tests for every high and medium finding

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
VF-01	Read-only reentrancy lets flash-loan callbacks liquidate vToken collateral at a depressed exchange rate	■ High	Read-only reentrancy	Resolved
VF-02	Oracle falls back to a manipulable AMM spot price whenever the primary feed exceeds a one-hour staleness window	■ High	Oracle manipulation	Resolved
VF-03	vToken permit domain separator uses a hard-coded chain ID and omits the verifying contract	■ Medium	Signature replay	Resolved
VF-04	Front-running the permit inside depositWithPermit blocks the deposit	■ Low	Denial of service	Resolved
VF-05	No grace period after the Base sequencer recovers before prices are used for liquidations	■ Medium	Oracle manipulation	Resolved
VF-06	Reserve-manager role remains with the deployment key	■ Medium	Access control	Resolved
VF-07	Share conversion rounds in the depositor's favour on mint and redeem	■ Low	Precision and rounding	Partially resolved
VF-08	Liquidation events are emitted before state changes and lack indexed fields	■ Informational	Code quality	Acknowledged

7

Detailed findings

VF-01 ■ High Resolved

Read-only reentrancy lets flash-loan callbacks liquidate vToken collateral at a depressed exchange rate

src/LendingPool.sol:L402-L447 (flashLoan), src/tokens/VToken.sol:L118 (exchangeRateStored), src/liquidation/Liquidator.sol:L86

Category	CWE	Likelihood	Impact
Read-only reentrancy	CWE-1265	Medium	High

DESCRIPTION

The vToken exchange rate is computed as $(\text{cash} + \text{totalBorrows} - \text{reserves}) / \text{totalSupply}$, where cash is the pool's token balance for the market. `flashLoan()` transfers the requested amount to the receiver and then calls `onFlashLoan()` before the funds come back. For the duration of the callback cash is lower by the loan amount while nothing else changes, so `exchangeRateStored()` reports a value that can be far below the true rate.

All state-changing pool functions share a `nonReentrant` guard, which correctly prevents a deposit or redemption at the wrong rate. The view function is not guarded, however, and the `Liquidator` is a separate contract with its own guard. When an account uses vTokens as collateral, `Liquidator.liquidate()` values that collateral with `exchangeRateStored()`, so a liquidation triggered from inside the callback sees the depressed value.

This is a read-only reentrancy: the pool never re-enters itself, but another contract reads the pool's state while it is temporarily inconsistent and acts on it.

PRECONDITIONS

- At least one market accepts vTokens as collateral (enabled for vUSDC and vWETH at the reviewed commit).
- Accounts holding vToken collateral are within the reach of the price drop the attacker can induce with available flash-loan liquidity.
- The flash-loaned market holds enough cash for the loan to move the exchange rate materially.

EXPLOIT SCENARIO

1. The attacker identifies accounts whose vUSDC collateral would fall below the liquidation threshold if the vUSDC exchange rate dropped by 15%.
2. The attacker flash-borrows most of the USDC market's cash, which lowers cash and therefore `exchangeRateStored()` for vUSDC.
3. Inside `onFlashLoan()`, the attacker calls `Liquidator.liquidate()` on each target. The liquidator reads the depressed rate, judges the accounts unhealthy and sells their collateral to the attacker at the liquidation discount.
4. The attacker repays the flash loan plus fee. The exchange rate returns to normal and the attacker now holds collateral bought at a discount from accounts that were never unhealthy.

IMPACT

Healthy borrowers lose the liquidation bonus on collateral that should never have been liquidated. Profit scales with the vToken collateral near the threshold and is limited only by flash-loan liquidity. Integrators outside Veyra that price vTokens with the same view function are exposed to the same manipulation.

RECOMMENDATION

Make the pool's accounting consistent during flash loans by adding outstanding flash-loan principal back into the asset

total, and make vToken valuation views revert while the pool is entered. The Liquidator should call a pool function that asserts the guard is not entered before valuing any collateral.

FIX

src/LendingPool.sol and src/tokens/VToken.sol (remediation commit)	DIFF
1	function flashLoan(address receiver, address asset, uint256 amount, bytes calldata data) external nonReentrant {
2	Market storage m = markets[asset];
3 +	m.flashOutstanding += amount;
4	IERC20(asset).safeTransfer(receiver, amount);
5	uint256 fee = _flashFee(amount);
6	require(IFlashBorrower(receiver).onFlashLoan(msg.sender, asset, amount, fee, data) == CALLBACK_SUCCESS, "callback");
7	IERC20(asset).safeTransferFrom(receiver, address(this), amount + fee);
8 +	m.flashOutstanding -= amount;
9	m.reserves += fee;
10	}
11	
12	function totalAssets(address asset) public view returns (uint256) {
13	Market storage m = markets[asset];
14 -	return m.cash() + m.totalBorrows - m.reserves;
15 +	return m.cash() + m.flashOutstanding + m.totalBorrows - m.reserves;
16	}
17	
18 +	/// Reverts if called while a state-changing pool function is executing.
19 +	function assertNotEntered() public view {
20 +	if (_reentrancyGuardEntered()) revert ReentrantRead();
21 +	}

RESOLUTION

Fixed. Outstanding flash-loan principal is included in totalAssets, exchangeRateStored() calls assertNotEntered(), and the Liquidator calls it before valuing collateral. A regression test reproduces the original attack and now reverts.

REFERENCES

- SWC-107: Reentrancy
<https://swcregistry.io/docs/SWC-107/>
- OpenZeppelin Contracts: ReentrancyGuard (utilities)
<https://docs.openzeppelin.com/contracts/5.x/api/utils>
- ERC-3156: Flash Loans
<https://eips.ethereum.org/EIPS/eip-3156>

VF-02 ■ High Resolved

Oracle falls back to a manipulable AMM spot price whenever the primary feed exceeds a one-hour staleness window

src/oracle/PriceOracle.sol:L71-L118 (getPrice, _fallbackPrice)

Category	CWE	Likelihood	Impact
Oracle manipulation	CWE-807	Medium	High

DESCRIPTION

getPrice(asset) reads the primary feed and, if block.timestamp - updatedAt > STALENESS (a global constant of 3,600 seconds), calls _fallbackPrice(asset), which derives the price from the reserves of a configured AMM pool: reserveQuote * 1e18 / reserveBase.

Primary feeds update on a deviation threshold or on a heartbeat. For assets that move slowly, the heartbeat is commonly 24 hours, so the feed is routinely more than an hour old while still correct. For those assets the fallback is active most of the day, and the fallback is a spot price that can be moved arbitrarily within a single transaction.

The flaw is the combination of a staleness rule that does not match the feed's heartbeat and a fallback source with no manipulation resistance. Either alone would be a lower-severity configuration issue.

PRECONDITIONS

- A listed collateral asset whose primary feed heartbeat is longer than one hour (two such assets at the reviewed commit).
- An AMM pool configured as the fallback source with liquidity small enough to be moved with flash-loan capital.

EXPLOIT SCENARIO

1. The attacker waits until the primary feed for the collateral asset is more than an hour old, which happens daily for a 24-hour heartbeat feed.
2. The attacker flash-borrows the quote asset and buys the collateral asset in the fallback pool, inflating its spot price several times.
3. The attacker deposits a small amount of the collateral asset into Veyra, which now values it at the inflated fallback price, and borrows other assets against it.
4. The attacker sells back into the pool, repays the flash loan and keeps the borrowed assets. The position is left with bad debt.

IMPACT

Depositors in the markets the attacker borrows from absorb the bad debt. The loss is bounded by the borrowable liquidity across markets and the collateral factor of the manipulated asset.

RECOMMENDATION

Remove the spot-price fallback. Configure staleness per feed, slightly above the feed's documented heartbeat, and when the primary price is stale stop new borrows and withdrawals of collateral priced by that feed rather than substituting another source. If a secondary source is required, use a time-weighted average over a window long enough to make manipulation uneconomic and bound the deviation between sources.

FIX

src/oracle/PriceOracle.sol (remediation commit)	DIFF
1 - uint256 public constant STALENESS = 1 hours;	
2 + mapping(address => uint256) public maxStaleness; // set per feed: heartbeat + buffer	
3	
4 function getPrice(address asset) public view returns (uint256) {	
5 (, int256 answer,, uint256 updatedAt,) = feeds[asset].latestRoundData();	
6 if (answer <= 0) revert InvalidPrice(asset);	
7 - if (block.timestamp - updatedAt > STALENESS) return _fallbackPrice(asset);	
8 + if (block.timestamp - updatedAt > maxStaleness[asset]) revert StalePrice(asset);	
9 return _scale(uint256(answer), feeds[asset].decimals());	
10 }	
11 -	
12 - function _fallbackPrice(address asset) internal view returns (uint256) {	
13 (uint112 r0, uint112 r1,) = fallbackPools[asset].getReserves();	
14 - return uint256(r1) * 1e18 / r0;	
15 - }	

RESOLUTION

Fixed. The fallback was removed and staleness is configured per feed. Borrowing against an asset whose price is stale now reverts; repayments and deposits remain possible.

REFERENCES

- Chainlink Data Feeds documentation
<https://docs.chain.link/data-feeds>
- Uniswap v3 price oracles
<https://developers.uniswap.org/docs/protocols/v3/concepts/price-oracles>

VF-03 ■ Medium Resolved

vToken permit domain separator uses a hard-coded chain ID and omits the verifying contract

src/tokens/VToken.sol:L44-L61 (DOMAIN_SEPARATOR, permit)

Category	CWE	Likelihood	Impact
Signature replay	CWE-294	Low	High

DESCRIPTION

The vToken implements EIP-2612 `permit` with its own domain separator: `keccak256(abi.encode(TYPEHASH, keccak256(bytes(name())), keccak256("1"), 8453))`. The chain ID is a literal and the `verifyingContract` field required by the EIP-712 domain recommendations is missing.

The Veyra team plans to deploy the same contracts to a second chain using deterministic deployment, so vTokens will have the same name and address there. Without a chain-specific value in the domain, a permit signed for a vToken on Base is equally valid for the vToken with the same name on the second deployment. Omitting the verifying contract also means two vTokens that ever share a name would accept each other's permits.

The nonce does not prevent this: nonces are tracked per contract, so the same signature with nonce `n` is fresh on each deployment.

PRECONDITIONS

- A second deployment of vTokens with the same name, which is on the Veyra roadmap.
- A holder who has signed a permit on one deployment and holds vTokens on the other.

EXPLOIT SCENARIO

- A user signs a permit on Base allowing a router to spend 1,000 vUSDC.
- An observer copies the signature from the Base transaction and submits it to the vUSDC contract on the second chain.
- The router (or anyone it lets act) can now move up to 1,000 vUSDC of the same user on the second chain without their consent there.

IMPACT

Allows the user never granted on one chain become usable by the spender named in the permit. The spender is chosen by the signer, which limits the attack to signatures naming a spender that can be abused, but routers and aggregators commonly expose such paths.

RECOMMENDATION

Use an EIP-712 implementation that builds the domain from `block.chainid` and `address(this)`, caches it, and rebuilds it if the chain ID changes after a fork.

FIX

src/tokens/VToken.sol (remediation commit)

DIFF

```
1 -contract VToken is ERC20 {
2 -    bytes32 public immutable DOMAIN_SEPARATOR =
3 -        keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(_name)), keccak256("1"), 8453));
4 +contract VToken is ERC20, ERC20Permit {
5 +    constructor(string memory name_, string memory symbol_) ERC20(name_, symbol_) ERC20Permit(n
6 +        ame_) {}
7 +    // ERC20Permit uses OpenZeppelin EIP712: domain includes block.chainid and address(this)
8 +    // and is recomputed if block.chainid differs from the cached value.
```

RESOLUTION

Fixed by adopting ERC20Permit from OpenZeppelin Contracts 5.x. Signatures from the original domain are no longer accepted.

REFERENCES

- EIP-712: Typed structured data hashing and signing
<https://eips.ethereum.org/EIPS/eip-712>
- EIP-2612: Permit extension for EIP-20 signed approvals
<https://eips.ethereum.org/EIPS/eip-2612>
- SWC-121: Missing protection against signature replay attacks
<https://swcregistry.io/docs/SWC-121/>

VF-04 ■ Low Resolved

Front-running the permit inside depositWithPermit blocks the deposit

src/LendingPool.sol:L188-L203 (depositWithPermit)

Category	CWE	Likelihood	Impact
Denial of service	CWE-400	Medium	Low

DESCRIPTION

`depositWithPermit()` calls `IERC20Permit(asset).permit(...)` and then pulls the tokens. The permit signature is visible in the public mempool before inclusion. Anyone can copy it and call `permit()` on the token directly first. The signature is then consumed, the nonce has advanced, and the pool's own `permit()` call reverts, failing the whole deposit.

No funds are at risk: the front-runner only sets the allowance the user intended to set. The effect is a grieving denial of service that costs the attacker gas and the user a failed transaction.

PRECONDITIONS

- The user deposits through `depositWithPermit` rather than a prior `approve`.

EXPLOIT SCENARIO

1. A user submits `depositWithPermit` with a valid permit signature.
2. A griever copies the signature and submits `permit()` directly to the token with a higher priority fee.
3. The user's transaction reverts on the already-used nonce.

IMPACT

Deposits can be blocked repeatedly for users relying on permits. No loss of funds.

RECOMMENDATION

Wrap the `permit()` call in `try/catch` and proceed if the allowance is already sufficient.

FIX

src/LendingPool.sol (remediation commit)	DIFF
1 - IERC20Permit(asset).permit(msg.sender, address(this), amount, deadline, v, r, s);	
2 + try IERC20Permit(asset).permit(msg.sender, address(this), amount, deadline, v, r, s) {} c	
atch {	
3 + if (IERC20(asset).allowance(msg.sender, address(this)) < amount) revert PermitFailed(	
);	
4 + }	
5 _deposit(asset, msg.sender, amount);	

RESOLUTION

Fixed as recommended.

REFERENCES

- EIP-2612: Security considerations
<https://eips.ethereum.org/EIPS/eip-2612>

VF-05 ■ Medium Resolved

No grace period after the Base sequencer recovers before prices are used for liquidations

src/oracle/PriceOracle.sol:L52-L66 (_checkSequencer)

Category	CWE	Likelihood	Impact
Oracle manipulation	CWE-754	Low	High

DESCRIPTION

The oracle reads the L2 sequencer-uptime feed and rejects prices while `answer == 1` (sequencer down). As soon as the feed reports the sequencer up again, prices are accepted.

While the sequencer is down, users cannot top up collateral or repay debt on L2, although prices can move on other venues. When the sequencer restarts, the first transactions to land can liquidate positions that became unhealthy during the outage before their owners had any chance to react. The feed exposes `startedAt` for the current status precisely so that integrators can wait a grace period.

PRECONDITIONS

- A sequencer outage during which collateral prices move against borrowers.

EXPLOIT SCENARIO

1. The sequencer goes down for 40 minutes while ETH falls 9%.
2. When it restarts, a liquidator with transactions queued through the forced-inclusion path or submitted first liquidates accounts that would have been topped up.

IMPACT

Borrowers lose liquidation bonuses they could have avoided. The impact is concentrated around outages, which are rare but correlated with volatile markets.

RECOMMENDATION

Reject prices until `block.timestamp - startedAt` exceeds a grace period (for example one hour) after the sequencer reports up, while still allowing repayments and deposits.

FIX

src/oracle/PriceOracle.sol (remediation commit)	DIFF
1 function _checkSequencer() internal view {	
2 - (, int256 answer,,,) = sequencerFeed.latestRoundData();	
3 + (, int256 answer, uint256 startedAt,,) = sequencerFeed.latestRoundData();	
4 if (answer != 0) revert SequencerDown();	
5 + if (block.timestamp - startedAt <= GRACE_PERIOD) revert GracePeriodNotOver();	
6 }	

RESOLUTION

Fixed with a one-hour grace period. Liquidations and new borrows revert during the grace period; repayments are allowed.

REFERENCES

- Chainlink: L2 sequencer uptime feeds
<https://docs.chain.link/data-feeds/l2-sequencer-feeds>

VF-06 ■ Medium Resolved

Reserve-manager role remains with the deployment key

script/Deploy.s.sol:L77, src/Treasury.sol:L33 (withdrawReserves)

Category	CWE	Likelihood	Impact
Access control	CWE-284	Low	High

DESCRIPTION

Treasury.withdrawReserves() is restricted to RESERVE_MANAGER_ROLE. The deployment script grants this role to the timelock and to msg.sender (the deployer) to configure initial reserve factors, and never revokes the deployer's copy. The deployment key is a hot key used by CI.

The contract logic is correct; the issue is the deployed role assignment, which the review covered because the deployment script was in scope. A compromise of the CI key would allow immediate withdrawal of all accrued reserves without passing through the timelock.

PRECONDITIONS

- Compromise of the deployment key.

EXPLOIT SCENARIO

- An attacker who obtains the CI deployment key calls withdrawReserves(asset, attacker, amount) for each market.

IMPACT

Loss of all protocol reserves, which also serve as the first loss buffer for bad debt.

RECOMMENDATION

Revoke every role from the deployer at the end of the deployment script and add a post-deployment check that only the timelock holds privileged roles.

FIX

script/Deploy.s.sol (remediation commit)	DIFF
1 treasury.grantRole(RESERVE_MANAGER_ROLE, address(timelock));	
2 - treasury.grantRole(RESERVE_MANAGER_ROLE, msg.sender);	
3 + // initial configuration performed through the timelock proposal batch	
4 + treasury.renounceRole(DEFAULT_ADMIN_ROLE, msg.sender);	

```
5 + require(!treasury.hasRole(RESERVE_MANAGER_ROLE, msg.sender), "deployer role");
```

RESOLUTION

Fixed. The role was revoked on-chain and the deployment script now asserts the final role set.

REFERENCES

- OpenZeppelin Contracts: Access control
<https://docs.openzeppelin.com/contracts/5.x/access-control>

VF-07 ■ Low Partially resolved

Share conversion rounds in the depositor's favour on mint and redeem

src/libraries/WadRayMath.sol:L41 (rayDiv), src/tokens/VToken.sol:L131-L150

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

DESCRIPTION

Both `mint` and `redeem` convert between underlying amounts and `VToken` shares with `rayDiv`, which rounds half up. Rounding should always favour the pool: shares minted on deposit should round down, and shares burned for a given withdrawal should round up.

Each operation leaks at most one unit of the smallest denomination, which is negligible for 18-decimal assets but adds up for 6-decimal assets across many small operations, and it breaks invariant tests that compare total assets with redeemable value.

PRECONDITIONS

- Any deposit or withdrawal whose conversion is not exact.

EXPLOIT SCENARIO

- A user repeatedly deposits and withdraws amounts chosen so that each conversion rounds up in their favour by one unit.

IMPACT

Value leaks from the pool at most one unit per operation; gas costs make deliberate extraction unprofitable.

RECOMMENDATION

Use explicit rounding direction: `mulDiv` with floor on mint and ceil on the shares burned for a withdrawal.

FIX

	src/tokens/VToken.sol (remediation commit)	DIFF
1 -	<code>uint256 shares = amount.rayDiv(rate);</code>	
2 +	<code>uint256 shares = Math.mulDiv(amount, RAY, rate, Math.Rounding.Floor);</code>	

RESOLUTION

Mint rounding fixed. Redeem still rounds half up; the team accepted the residual one-unit leakage pending a broader math refactor.

REFERENCES

- EIP-4626: Tokenized Vaults (rounding direction)
<https://eips.ethereum.org/EIPS/eip-4626>
- CWE-682: Incorrect Calculation
<https://cwe.mitre.org/data/definitions/682.html>

VF-08 ■ Informational Acknowledged

Liquidation events are emitted before state changes and lack indexed fields

src/liquidation/Liquidator.sol:L141, L188

Category	CWE	Likelihood	Impact
Code quality	—	Low	Low

DESCRIPTION

Liquidated and AuctionSettled are emitted before the collateral transfer and debt update, and neither indexes the borrower or the liquidator. Off-chain monitoring has to decode every event to find liquidations of a specific account, and a revert after the emit in a future refactor could mislead indexers that read pending transactions.

PRECONDITIONS

- None; affects observability only.

EXPLOIT SCENARIO

1. Not exploitable.

IMPACT

Slower and less reliable monitoring of liquidations.

RECOMMENDATION

Emit after state changes and index `borrower`, `liquidator` and `market`.

RESOLUTION

Acknowledged. Planned for the next release because indexers already depend on the current event signature.

REFERENCES

- Solidity documentation: Events
<https://docs.soliditylang.org/en/latest/contracts.html#events>
- Solidity documentation: Security considerations
<https://docs.soliditylang.org/en/latest/security-considerations.html>

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

	Low	Medium	High
High	3 VF-03 VF-05 VF-06	2 VF-01 VF-02	0
Medium	0	0	0
Low	1 VF-08	2 VF-04 VF-07	0
	Low	Medium	High

Likelihood

Distribution by vulnerability class

Oracle manipulation	2
Access control	1
Code quality	1
Denial of service	1
Precision and rounding	1
Read-only reentrancy	1
Signature replay	1

Severity distribution



9

Remediation status

6 of 8 findings are resolved, 1 acknowledged, 1 partially resolved and 0 open, as reviewed at remediation commit d04a8f2c6b19.

Severity	Resolved	Partially resolved	Acknowledged	Open	Total
■ Critical	0	0	0	0	0
■ High	2	0	0	0	2
■ Medium	3	0	0	0	3
■ Low	1	1	0	0	2
■ Informational	0	0	1	0	1

Remediation progress

94%

Remediation history



Residual items

ID	Status	Notes
VF-07	Partially resolved	Mint rounding fixed. Redeem still rounds half up; the team accepted the residual one-unit leakage pending a broader math refactor.
VF-08	Acknowledged	Acknowledged. Planned for the next release because indexers already depend on the current event signature.

10

Formal verification and invariants

Four accounting properties were expressed as bounded symbolic tests and run against the remediation commit. The asset-conservation property failed on the original commit while a flash loan was outstanding, which is how VF-01 was first confirmed; it holds after the fix. Monotonicity of the exchange rate could not be completed within the solver time budget.

Tool	Halmos (bounded symbolic execution) and Foundry invariant tests
------	---

ID	Property	Result	Notes
P1	Pool assets used for the exchange rate are never less than cash + borrows – reserves, including while a flash loan is outstanding.	■ Proved	Violated on the original commit (VF-01); holds on the remediation commit up to depth 6.
P2	A successful borrow never leaves the borrower with a health factor below 1.	■ Proved	Bounded to three markets and two collateral assets.
P3	A flash loan either returns principal plus fee to the pool or reverts.	■ Proved	–
P4	The vToken exchange rate never decreases except through bad-debt realisation in a liquidation.	■ Inconclusive	Solver timeout at depth 5; no counterexample found by 50 million fuzzing runs.

11

Recommendations

Keep vToken valuation behind the reentrancy check

Any new contract or integration that values vTokens must call `assertNotEntered()` or read the guarded view. Add this to the integration guide.

Document feed heartbeats with each listing

Record the primary feed heartbeat and deviation threshold for every listed asset and set `maxStaleness` from it in the same governance proposal.

Run the invariant suite in CI

The invariant and symbolic tests written during the review reproduce both high-severity findings. Running them on every change protects the fixes.

Assert final role assignments after every deployment

Extend the deployment script checks to every privileged role on every contract, including future chains.

Monitor oracle and sequencer status

Alert on stale-price reverts and on sequencer status changes so that operators know when borrowing is paused.

12

Limitations

- Only the files and commit listed in scope were reviewed; later changes are not covered.
- Price-feed providers and the Base sequencer were treated as behaving as documented.
- Economic analysis assumed liquidity conditions at the time of review; thinner liquidity increases manipulation risk.
- Bounded symbolic tests prove properties only up to the stated depth and market count.

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
Foundry	Unit, fork and invariant tests; proof-of-concept exploits
Slither	Static analysis
Halmos	Bounded symbolic tests for accounting properties
Custom scripts	Role-assignment extraction from deployment broadcasts

A.2 Reviewed files (SHA-256)

File	SHA-256
src/LendingPool.sol	adeccdfa16289b6c40e8513c2a36d16449eb14c46b0fd465c1db7bdba7f6096d
src/tokens/VToken.sol	f658c9fb3c4b4f0109267c68b00a7551cdee7e8553150b1c51cf413554f8f1f4
src/oracle/PriceOracle.sol	56f616f1a90f2f59f81b55cb9117fdd1afac8757e036a95baa63cd6967a7c28a
src/liquidation/Liquidator.sol	0d4d579accd5fe21164d6a553c3fd0e418bf66d162c3b01411afffd724f57cfc
src/Treasury.sol	8ad9f6df56917d2239f25e1fcdbd1046d176d146f2b3c591f808f7e8d6778d1a8
src/libraries/WadRayMath.sol	9019bc6a83f3f32542a04575d1b99acdf786ced721ddbfb2dfbd29c8bb7351db1

A.3 Notes

- File hashes are SHA-256 digests of the reviewed sources at the audited commit.
- Severity follows the Quantix impact × likelihood matrix described in the methodology.

A.4 Methodology and technical references

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

1. SWC-107: Reentrancy
<https://swcregistry.io/docs/SWC-107/>
2. OpenZeppelin Contracts: ReentrancyGuard (utilities)
<https://docs.openzeppelin.com/contracts/5.x/api/utils>
3. ERC-3156: Flash Loans
<https://eips.ethereum.org/EIPS/eip-3156>
4. Chainlink Data Feeds documentation
<https://docs.chain.link/data-feeds>
5. Uniswap v3 price oracles
<https://developers.uniswap.org/docs/protocols/v3/concepts/price-oracles>
6. EIP-712: Typed structured data hashing and signing
<https://eips.ethereum.org/EIPS/eip-712>
7. EIP-2612: Permit extension for EIP-20 signed approvals
<https://eips.ethereum.org/EIPS/eip-2612>
8. SWC-121: Missing protection against signature replay attacks
<https://swcregistry.io/docs/SWC-121/>

9. Chainlink: L2 sequencer uptime feeds
<https://docs.chain.link/data-feeds/l2-sequencer-feeds>
10. OpenZeppelin Contracts: Access control
<https://docs.openzeppelin.com/contracts/5.x/access-control>
11. EIP-4626: Tokenized Vaults (rounding direction)
<https://eips.ethereum.org/EIPS/eip-4626>
12. CWE-682: Incorrect Calculation
<https://cwe.mitre.org/data/definitions/682.html>
13. Solidity documentation: Events
<https://docs.soliditylang.org/en/latest/contracts.html#events>
14. Solidity documentation: Security considerations
<https://docs.soliditylang.org/en/latest/security-considerations.html>

A.5 Document information

Document	Quantix Technical Security Assessment QX-TSA-2026-007, version 1.1
Subject	Veyra Finance, Quantix Protocol Research
Status	Published 24 March 2026
Disclosure	This document is a Quantix research assessment of Veyra Finance, 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.