



TECHNICAL SECURITY ASSESSMENT

Obsidian Rollup

Protocol Architecture Review · Layer 2

REPORT ID QX-TSA-2026-008

QUANTIX RESEARCH ASSESSMENT

L1 rollup contracts (batch inbox, output oracle, dispute game, portal), the v2 upgrade and the Go sequencer and batcher. 4,310 nSLOC.

Document status
Published

Published
5 February 2026

Review period
1 December 2025 – 9 January 2026

Ecosystems
Ethereum

Languages
Solidity, Go

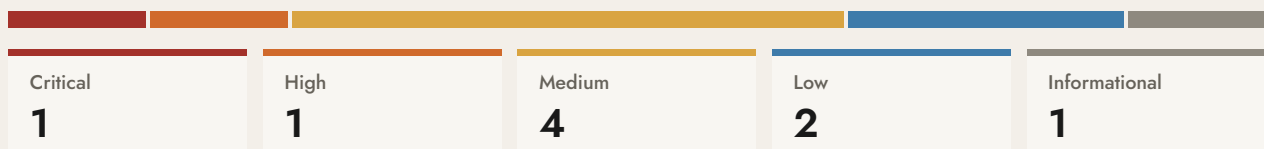
Review effort
Four reviewers over twenty-five working days, including the pending v2 upgrade and the Go sequencer

Reviewed commit
2f9b6e01c4a87d35e9f0b12c6a4d8e73f1a05b96

Remediation commit
8a3d5c7e19f02b64d8e1a9c0f37b25e64c0d1a73

FINDINGS BY SEVERITY

9 findings · 94% remediated



This document is a Quantix research assessment of Obsidian Rollup, 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-2026-008
Title	Technical Security Assessment: Obsidian Rollup
Version	1.2
Status	Published
Assessment type	Protocol Architecture Review
Review period	1 December 2025 – 9 January 2026
Published	5 February 2026
Reviewed commit	2f9b6e01c4a87d35e9f0b12c6a4d8e73f1a05b96

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
	OR-01 OutputOracle v2 inserts a storage variable before existing slots, corrupting the finalisation period on...	8
	OR-02 L1 contracts can be upgraded without delay by the operations multisig	9
	OR-03 Withdrawals can be finalised against a dispute game that has not resolved for the defender	10
	OR-04 Withdrawals can be forced to fail with insufficient gas and are then replayable at a time of anyone's...	11
	OR-05 Batcher does not detect L1 reorganisations of submitted batches, stalling the safe head	12
	OR-06 Deposits are not priced, allowing cheap spam that fills L2 blocks with forced transactions	13
	OR-07 Challenger role can delete finalised outputs	14
	OR-08 Batcher logs its configuration, including the signer key path and RPC credentials, at debug level	15
	OR-09 SystemConfig setters do not emit dedicated events	16
8	Severity and risk analysis	17
9	Remediation status	18
10	Formal verification and invariants	19
11	Recommendations	19
12	Limitations	20
A	Technical appendix	21

1

Executive summary

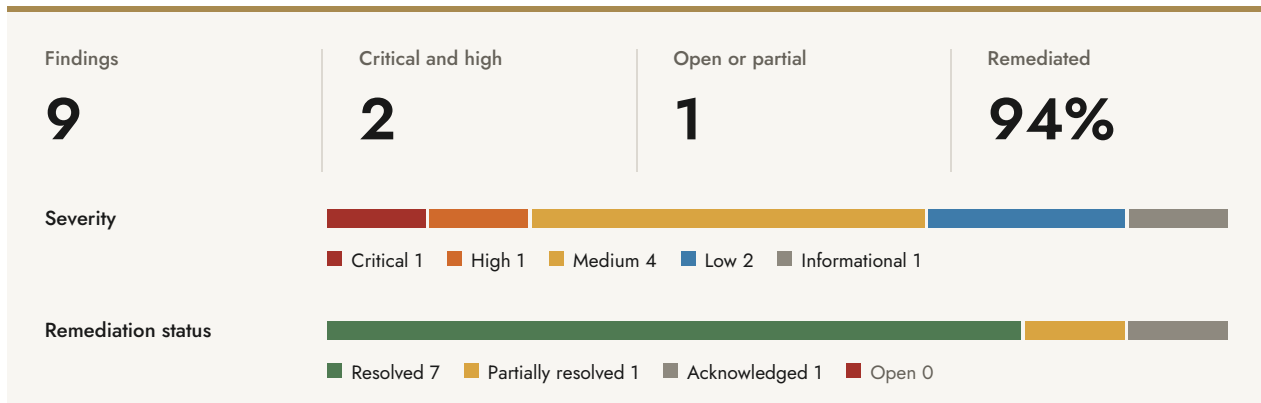
Obsidian Rollup is an Ethereum Layer 2 that posts transaction batches to L1, proposes L2 output roots, and settles withdrawals through a fault-proof dispute game after a challenge window. A single operator runs the sequencer; users can force-include deposits through the L1 Portal. Quantix reviewed the L1 contracts, the pending v2 upgrade, and the Go batcher at the commits listed in this report.

The most serious issue was in the v2 upgrade rather than the deployed system: the new `OutputOracle` implementation inserted a storage variable ahead of existing ones. Executing the upgrade would have shifted every later slot, making the contract read the stored output count as the finalisation period and allowing withdrawals to be finalised without a challenge window. This critical finding was fixed before the upgrade was proposed.

A high-severity issue allowed withdrawals to be finalised against a dispute game that had not yet resolved in the proposer's favour, provided the proof-maturity delay had passed. Four medium-severity findings covered upgrade governance, gas griefing of withdrawals, batch resubmission after L1 reorganisations, and unpriced deposit spam. All findings were resolved except the upgrade-delay recommendation, which was partially adopted.

The system remains in a permissioned stage: a single sequencer orders transactions and a security council can upgrade contracts without delay in an emergency. These properties are documented and are reflected in the governance component of the protocol's risk profile.

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.	4	1
■ Low	Minor impact, defence-in-depth gaps, or issues with unlikely preconditions.	2	0

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

2

Scope

L1 rollup contracts (batch inbox, output oracle, dispute game, portal), the v2 upgrade and the Go sequencer and batcher. 4,310 nSLOC.

Repository	github.com/example-org/obsidian-rollup
Reviewed commit	2f9b6e01c4a87d35e9f0b12c6a4d8e73f1a05b96
Remediation commit	8a3d5c7e19f02b64d8e1a9c0f37b25e64c0d1a73
Review period	1 December 2025 — 9 January 2026
Review effort	Four reviewers over twenty-five working days, including the pending v2 upgrade and the Go sequencer

Path	Description	nSLOC
contracts/L1/BatchInbox.sol	Receives compressed transaction batches from the batcher	142
contracts/L1/OutputOracle.sol	Stores proposed L2 output roots; v1 and pending v2	318
contracts/dispute/DisputeGame.sol	Bisection fault-proof game over disputed outputs	1,046
contracts/L1/Portal.sol	Deposits, withdrawal proving and finalisation	684
contracts/L1/SystemConfig.sol	Batcher, gas and resource parameters	221
contracts/L1/ResourceMetering.sol	Deposit gas pricing (added during remediation)	167
op-sequencer/batcher/*.go	Batch construction, submission and retry logic	1,238
script/UpgradeV2.s.sol	Proxy upgrade script for the v2 release	94
Total		3,910

Out of scope

- The L2 execution client and derivation pipeline beyond batch submission
- Fault-proof VM and off-chain challenger agent
- L2 predeploys other than the withdrawal message passer interface
- Operational security of the security council signers

3

Protocol architecture

The batcher compresses L2 transactions into frames and posts them to `BatchInbox` on L1. A proposer periodically submits output roots to `OutputOracle`; each output can be challenged by opening a `DisputeGame`, a bisection game that narrows a disagreement to a single instruction executed by an on-chain fault-proof VM.

Withdrawals are initiated on L2, proven on L1 against an output root through `Portal.proveWithdrawalTransaction`, and finalised after the proof has matured and the corresponding game has resolved. Deposits enter through `Portal.depositTransaction` and must be included by the sequencer within a sequencing window, or they are force-included by derivation.

All L1 contracts sit behind EIP-1967 proxies administered by a `ProxyAdmin` owned by a 2-of-3 operations multisig, with a 4-of-7 security council able to act in emergencies.

Component	Role	Trust level
<code>BatchInbox</code>	Data availability for L2 batches	Trusted
<code>OutputOracle</code>	Proposed L2 state roots	Trusted
<code>DisputeGame</code>	Permissionless fault proofs	Trusted
<code>Portal</code>	Deposits and withdrawals; holds bridged ETH	Trusted
Sequencer and batcher (Go)	Transaction ordering and batch posting	Semi-trusted
Proposer	Submits output roots	Semi-trusted
Challengers, depositors, withdrawers	Any external account	Untrusted

Trust assumptions

- At least one honest challenger monitors outputs and responds within the game clock.
- Ethereum L1 provides data availability and censorship resistance for forced inclusion.
- The security council does not collude to push a malicious upgrade.
- The fault-proof VM executes L2 state transitions faithfully (out of scope).

4

Threat model

Assets

- ETH and tokens held by the Portal
- Correctness of finalised withdrawals
- Liveness of L2 block production
- Integrity of the upgrade path

Actors

Actor	Capability	Trust
Malicious proposer	Submits invalid output roots	Semi-trusted
Withdrawal griever	Finalises other users' withdrawals with chosen gas	Untrusted
Deposit spammer	Submits many deposits with large gas limits	Untrusted
Upgrade administrator	Upgrades L1 proxies	Trusted

Threats

Threat	Actor and vector	Mitigation	Findings
Corrupt the challenge period during an upgrade	Upgrade administrator: Storage layout shift in the v2 implementation	Append-only layout, namespaced storage, layout diff in CI	OR-01
Finalise a withdrawal backed by an invalid output	Malicious proposer: Finalise before the dispute game resolves	Require DEFENDER_WINS plus a finality delay	OR-03
Upgrade the Portal without notice	Upgrade administrator: Instant ProxyAdmin upgrade	Timelock for non-emergency upgrades	OR-02
Make withdrawals fail permanently	Withdrawal griever: Finalise with too little gas for the target call	Minimum-gas check before the external call	OR-04
Stall the safe head	Ethereum reorganisation: Batcher loses track of a reorged batch	Track inclusion by block hash and resubmit	OR-05
Fill L2 blocks with forced deposits	Deposit spammer: Unpriced deposit gas	EIP-1559-style resource metering	OR-06

5

Methodology and severity classification

1. Architecture walk-through and threat modelling with the Obsidian core team
2. Manual review of all in-scope Solidity and Go code by two reviewers per component
3. Storage-layout comparison of every v1 and v2 implementation
4. Differential tests of withdrawal proving and finalisation against the OP Stack specification behaviour
5. Symbolic tests of the dispute-game bisection bounds and Portal state machine
6. L1 reorganisation simulation against the batcher with a local devnet
7. Remediation review of the fix commit and of the revised upgrade script

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
OR-01	OutputOracle v2 inserts a storage variable before existing slots, corrupting the finalisation period on upgrade	■ Critical	Unsafe proxy upgrade	Resolved
OR-02	L1 contracts can be upgraded without delay by the operations multisig	■ Medium	Governance manipulation	Partially resolved
OR-03	Withdrawals can be finalised against a dispute game that has not resolved for the defender	■ High	Business logic	Resolved
OR-04	Withdrawals can be forced to fail with insufficient gas and are then replayable at a time of anyone's choosing	■ Medium	Denial of service	Resolved
OR-05	Batcher does not detect L1 reorganisations of submitted batches, stalling the safe head	■ Medium	Denial of service	Resolved
OR-06	Deposits are not priced, allowing cheap spam that fills L2 blocks with forced transactions	■ Medium	Denial of service	Resolved
OR-07	Challenger role can delete finalised outputs	■ Low	Access control	Resolved
OR-08	Batcher logs its configuration, including the signer key path and RPC credentials, at debug level	■ Low	Key management	Resolved
OR-09	SystemConfig setters do not emit dedicated events	■ Informational	Code quality	Acknowledged

7

Detailed findings

OR-01 ■ Critical Resolved

OutputOracle v2 inserts a storage variable before existing slots, corrupting the finalisation period on upgrade

contracts/L1/OutputOracle.sol (v2):L28-L41, script/UpgradeV2.s.sol:L52

Category	CWE	Likelihood	Impact
Unsafe proxy upgrade	CWE-664	High	High

DESCRIPTION

A proxy keeps its state in its own storage and executes the implementation's code against it. Every implementation must therefore preserve the slot assignment of existing variables; new variables may only be appended or placed in dedicated namespaces.

The v1 `OutputOracle` declares, in order, `startingBlockNumber`, `startingTimestamp`, `l2Outputs` (a dynamic array) and `finalizationPeriodSeconds` set during initialisation. The v2 implementation adds `uint256 public proposalBond` as the first state variable, before `startingBlockNumber`. After the upgrade every existing variable is read one slot later: `startingTimestamp` reads the old `startingBlockNumber`, the array length of `l2Outputs` is read from the old `startingTimestamp` slot, and `finalizationPeriodSeconds` reads the slot that previously held the length of `l2Outputs`.

Because `finalizationPeriodSeconds` would equal the number of outputs proposed so far (a few thousand), the seven-day challenge window would shrink to roughly an hour. The upgrade script calls `upgradeToAndCall` with no layout validation, so the shift would have taken effect on execution.

PRECONDITIONS

- The v2 upgrade is executed as scripted.

EXPLOIT SCENARIO

1. The operations multisig executes `UpgradeV2.s.sol`.
2. The `OutputOracle` now reports a finalisation period equal to the old output count, about 3,400 seconds.
3. A malicious proposer submits an invalid output root, proves a withdrawal of the Portal's ETH against it, and finalises after an hour, before honest challengers can complete a dispute game.

IMPACT

Loss of bridged funds held by the Portal and invalid L2 state accepted on L1. The finding was caught before the upgrade was proposed; no deployed contract was affected.

RECOMMENDATION

Append new variables after existing ones or move state into an EIP-7201 namespaced struct. Add an automated storage-layout comparison between the deployed and new implementations to the upgrade pipeline and refuse to produce upgrade calldata when the check fails.

FIX

contracts/L1/OutputOracle.sol (v2, remediation commit)	DIFF
1	contract OutputOracle is Initializable, ISemver {
2	- uint256 public proposalBond;
3	uint256 public startingBlockNumber;
4	uint256 public startingTimestamp;

```

5      Types.OutputProposal[] internal l2Outputs;
6      uint256 public finalizationPeriodSeconds;
7  +    /// @custom:storage-location erc7201:obsidian.storage.OutputOracleV2
8  +    struct V2Storage { uint256 proposalBond; }
9  +    bytes32 private constant V2_SLOT = 0x6c1b2a4f63e0d2d4b5b2a5c6a0e9b8d3e7f1a2c4b6d8e0f2a4c6e8
    b0d2f4a600;

```

RESOLUTION

Fixed. New state lives in a namespaced struct, and the upgrade pipeline now diffs `forge inspect storageLayout` output against the deployed implementation and fails on any change to existing slots.

REFERENCES

- EIP-1967: Proxy storage slots
<https://eips.ethereum.org/EIPS/eip-1967>
- EIP-7201: Namespaced storage layout
<https://eips.ethereum.org/EIPS/eip-7201>
- OpenZeppelin: Writing upgradeable contracts
<https://docs.openzeppelin.com/upgrade-plugins/writing-upgradeable>

OR-02 ■ Medium Partially resolved

L1 contracts can be upgraded without delay by the operations multisig

contracts/universal/ProxyAdmin.sol (deployment configuration), contracts/L1/Portal.sol

Category	CWE	Likelihood	Impact
Governance manipulation	CWE-284	Low	High

DESCRIPTION

The `ProxyAdmin` that controls every L1 proxy is owned directly by a 2-of-3 operations multisig. An upgrade takes effect in the block it is executed. Users therefore have no window to withdraw if they disagree with an upgrade, and a compromise of two operations signers would give immediate control of all bridged funds.

Rollups at this stage commonly keep an emergency upgrade path, but separate it from routine upgrades, which pass through a public delay longer than the withdrawal period so that users can exit first.

PRECONDITIONS

- Two of three operations signers act maliciously or are compromised.

EXPLOIT SCENARIO

1. The compromised signers upgrade the Portal to an implementation that transfers its ETH balance to an attacker-controlled address.

IMPACT

Total loss of bridged assets in a key-compromise scenario. No defect in contract logic.

RECOMMENDATION

Route routine upgrades through a timelock longer than the withdrawal period (at least eight days) and reserve instant upgrades for the security council, with that power documented and monitored.

FIX

Deployment configuration (remediation)	DIFF
1 - proxyAdmin.owner = OPERATIONS_MULTISIG // 2-of-3, instant	
2 + proxyAdmin.owner = UpgradeTimelock // minDelay = 48 hours, proposer = OPERATIONS_MULTISIG	
3 + UpgradeTimelock.EMERGENCY_ROLE = SECURITY_COUNCIL // 4-of-7, instant, publicly logged	

RESOLUTION

Partially resolved. Routine upgrades now pass through a 48-hour `UpgradeTimeLock`, shorter than the eight days recommended, so users still cannot always exit before an upgrade takes effect; the 4-of-7 security council retains an instant path for emergencies, which Obsidian documents in its public risk disclosures.

REFERENCES

- OP Stack specifications
<https://specs.optimism.io/>

OR-03 ■ High Resolved

Withdrawals can be finalised against a dispute game that has not resolved for the defender

`contracts/L1/Portal.sol:L301-L337` (`finalizeWithdrawalTransaction`, `_checkWithdrawal`)

Category	CWE	Likelihood	Impact
Business logic	CWE-841	Medium	High

DESCRIPTION

When a withdrawal is proven, the Portal records the dispute game that backs the output root and the proof timestamp. `_checkWithdrawal` later verifies that `PROOF_MATURITY_DELAY` has passed since proving and that `game.status() != GameStatus.CHALLENGER_WINS`.

The check is phrased as "not lost" rather than "won". A game that is still `IN_PROGRESS` passes it. Dispute games can legitimately run longer than the proof-maturity delay when bisection is deep and both parties use their full clocks, so a withdrawal proven against an output that is under active dispute can be finalised before the game concludes. If the challenger later wins, the withdrawal has already been paid.

The OP Stack design that Obsidian follows requires the game to have resolved in favour of the defender and an additional "air-gap" delay after resolution, during which a guardian can blacklist a game resolved incorrectly.

PRECONDITIONS

- An invalid output root has been proposed and disputed.
- The dispute game runs longer than the proof-maturity delay.

EXPLOIT SCENARIO

- A malicious proposer submits an output root that includes a fabricated withdrawal of 5,000 ETH.
- An honest challenger opens a dispute game; the proposer plays every move at the last moment so the game stays in progress.
- After the proof-maturity delay, the proposer finalises the withdrawal. `_checkWithdrawal` passes because the game is not `CHALLENGER_WINS` yet.
- The game later resolves for the challenger, but the ETH has already left the Portal.

IMPACT

Theft of Portal funds by a malicious proposer despite an honest challenger, defeating the purpose of fault proofs.

RECOMMENDATION

Require `game.status() == GameStatus.DEFENDER_WINS` and `block.timestamp - game.resolvedAt() > DISPUTE_GAME_FINALITY_DELAY`, and let the guardian blacklist games during that delay.

FIX

contracts/L1/Portal.sol (remediation commit)	DIFF
1	function _checkWithdrawal(bytes32 withdrawalHash, ProvenWithdrawal memory p) internal view {
2	IDisputeGame game = p.disputeGameProxy;
3	- require(game.status() != GameState.CHALLENGER_WINS, "Portal: game lost");
4	+ require(!disputeGameBlacklist[game], "Portal: game blacklisted");
5	+ require(game.status() == GameState.DEFENDER_WINS, "Portal: game not won");
6	+ require(block.timestamp - game.resolvedAt().raw() > DISPUTE_GAME_FINALITY_DELAY, "Portal: finality delay");
7	require(block.timestamp - p.timestamp > PROOF_MATURITY_DELAY, "Portal: proof not mature")
8	;
	}

RESOLUTION

Fixed as recommended with a 3.5-day finality delay and a guardian blacklist. Differential tests against the specification behaviour pass.

REFERENCES

- OP Stack specifications: withdrawals and fault proofs
<https://specs.optimism.io/>
- CWE-841: Improper Enforcement of Behavioral Workflow
<https://cwe.mitre.org/data/definitions/841.html>

OR-04 ■ Medium Resolved

Withdrawals can be forced to fail with insufficient gas and are then replayable at a time of anyone's choosing

contracts/L1/Portal.sol:L338-L372 (finalizeWithdrawalTransaction)

Category	CWE	Likelihood	Impact
Denial of service	CWE-691	Medium	Medium

DESCRIPTION

Anyone may finalise any proven withdrawal. The Portal executes the withdrawal's target call with `gasleft()` and, if the call fails, leaves `finalizedWithdrawals[hash]` unset so that it can be retried.

Two problems follow. First, a griever can finalise someone else's withdrawal with just enough gas for the Portal's own logic but too little for the target call; because of the 63/64 rule the call fails while the outer transaction succeeds.

Second, the failed withdrawal remains open indefinitely and can be executed later by anyone, at a moment the original sender did not choose, for example after the target contract has changed state.

The specification addresses this by recording the withdrawal as finalised whether or not the call succeeds and by refusing to make the call unless enough gas remains to forward the withdrawal's declared gas limit.

PRECONDITIONS

- A proven withdrawal whose target call requires more than trivial gas.

EXPLOIT SCENARIO

1. A user proves a withdrawal that calls a vault on L1 to deposit the withdrawn ETH.
2. A griever calls `finalizeWithdrawalTransaction` with a gas limit chosen so the vault call runs out of gas.
3. The withdrawal is left unfinalised. Weeks later, after the vault has been paused and migrated, the griever finalises it again; the call now sends ETH to a deprecated contract.

IMPACT

Withdrawals fail or execute in unexpected states; funds can be stranded in contracts that no longer handle them.

RECOMMENDATION

Mark the withdrawal finalised before the call, and require `SafeCall.hasMinGas(gasLimit, RESERVED_GAS)` so that the declared gas limit is always forwarded.

FIX

	contracts/L1/Portal.sol (remediation commit)	DIFF
1	+	<code>finalizedWithdrawals[withdrawalHash] = true;</code>
2	+	<code>require(SafeCall.hasMinGas(_tx.gasLimit, RECEIVE_DEFAULT_GAS_LIMIT), "Portal: insufficient gas");</code>
3		<code>l2Sender = _tx.sender;</code>
4		<code>bool success = SafeCall.callWithMinGas(_tx.target, _tx.gasLimit, _tx.value, _tx.data);</code>
5		<code>l2Sender = Constants.DEFAULT_L2_SENDER;</code>
6	-	<code>if (success) finalizedWithdrawals[withdrawalHash] = true;</code>
7		<code>emit WithdrawalFinalized(withdrawalHash, success);</code>

RESOLUTION

Fixed. Withdrawals are final after one attempt and the declared gas is guaranteed to the target.

REFERENCES

- OP Stack specifications
<https://specs.optimism.io/>

OR-05 ■ Medium Resolved

Batcher does not detect L1 reorganisations of submitted batches, stalling the safe head

op-sequencer/batcher/tx_manager.go:L210-L268, op-sequencer/batcher/channel_manager.go:L144

Category	CWE	Likelihood	Impact
Denial of service	CWE-754	Medium	Medium

DESCRIPTION

The batcher records a frame as confirmed when its transaction receipt is returned and then prunes the frame from memory. Confirmation is checked once, at the first receipt. If the L1 block containing the batch is reorganised out, the transaction may never be re-included (the batcher has already moved its nonce forward and later transactions fill the gap), and the frame is gone.

Derivation on L2 cannot advance the safe head past the missing frame until the channel times out, after which the corresponding L2 blocks are re-derived without those transactions. In practice this halts finality for the duration of the channel timeout and can reorder user transactions.

PRECONDITIONS

- An L1 reorganisation deeper than one block affecting a block that contains a batch.

EXPLOIT SCENARIO

1. A two-block L1 reorg drops the block containing frame 3 of a channel.
2. The batcher has pruned frame 3 and keeps submitting frames 4 and 5.
3. Derivation waits for frame 3 until the channel times out; the safe head stalls for about 50 minutes.

IMPACT

Liveness degradation of L2 finality; bridges and exchanges relying on the safe head stop crediting deposits.

RECOMMENDATION

Track confirmations by block hash, keep frames until they are buried under a configurable confirmation depth, and requeue frames whose inclusion block is no longer canonical.

FIX

```

op-sequencer/batcher/tx_manager.go (remediation commit)
DIFF
1 - if receipt != nil {
2 -     m.channels.FrameConfirmed(frameID)
3 - }
4 + if receipt != nil {
5 +     m.pending[frameID] = inclusion{blockHash: receipt.BlockHash, blockNumber: receipt.BlockNumber.Uint64()}
6 + }
7 + // on each new L1 head:
8 + for id, inc := range m.pending {
9 +     if !m.l1.IsCanonical(ctx, inc.blockHash) {
10 +         m.channels.RequeueFrame(id)
11 +         delete(m.pending, id)
12 +     } else if head-inc.blockNumber >= m.cfg.ConfirmationDepth {
13 +         m.channels.FrameConfirmed(id)
14 +         delete(m.pending, id)
15 +     }
16 + }

```

RESOLUTION

Fixed. Frames are retained until ten confirmations and requeued when their block leaves the canonical chain; a devnet reorg test reproduces the original stall and passes after the fix.

REFERENCES

- OP Stack specification
<https://specs.optimism.io/>
- CWE-754: Improper Check for Unusual or Exceptional Conditions
<https://cwe.mitre.org/data/definitions/754.html>

OR-06 ■ Medium Resolved

Deposits are not priced, allowing cheap spam that fills L2 blocks with forced transactions

contracts/L1/Portal.sol:L120-L162 (depositTransaction)

Category	CWE	Likelihood	Impact
Denial of service	CWE-770	Medium	Medium

DESCRIPTION

Deposits submitted to the Portal must be included at the start of L2 blocks. The Portal checks only that the requested L2 gas limit is above a minimum derived from calldata size; nothing charges the depositor for the L2 gas they reserve.

An attacker can submit many deposits with large gas limits from L1, paying only L1 gas. Each deposit reserves L2 block gas, crowding out sequenced transactions and raising L2 fees for everyone. Designs derived from the OP Stack meter deposit gas with an EIP-1559-style market on L1, burning L1 gas proportional to the L2 gas purchased.

PRECONDITIONS

- None beyond L1 gas costs.

EXPLOIT SCENARIO

- An attacker submits 200 deposits per L1 block, each requesting 15 million L2 gas, saturating deposit capacity for several hours at modest L1 cost.

IMPACT

Degraded L2 throughput and elevated fees; sequenced transactions may be delayed.

RECOMMENDATION

Introduce resource metering for deposits: track L2 gas purchased per L1 block, adjust a base fee toward a target, and burn the corresponding L1 gas from the depositor.

FIX

contracts/L1/Portal.sol (remediation commit)	DIFF
<pre>1 - function depositTransaction(address _to, uint256 _value, uint64 _gasLimit, bool _isCreation, bytes memory _data) public payable { 2 + function depositTransaction(address _to, uint256 _value, uint64 _gasLimit, bool _isCreation, bytes memory _data) 3 + public payable metered(_gasLimit) 4 + {</pre>	

RESOLUTION

Fixed with a `ResourceMetering` base contract and parameters stored in `SystemConfig`.

REFERENCES

- EIP-1559: Fee market change
<https://eips.ethereum.org/EIPS/eip-1559>
- OP Stack specifications
<https://specs.optimism.io/>

OR-07 ■ Low Resolved

Challenger role can delete finalised outputs

contracts/L1/OutputOracle.sol:L188 (deleteL2Outputs)

Category	CWE	Likelihood	Impact
Access control	CWE-284	Low	Medium

DESCRIPTION

`deleteL2Outputs(index)` lets the CHALLENGER role remove outputs from `index` onward. It checks that the index exists but not that the outputs are still within the finalisation period. A mistaken or compromised challenger could delete finalised outputs, invalidating proofs for withdrawals that users have already proven and forcing them to re-prove against new outputs.

PRECONDITIONS

- Action by the trusted challenger role.

EXPLOIT SCENARIO

- The challenger key deletes outputs back to an index finalised two weeks earlier; pending withdrawals must be re-proven.

IMPACT

Delayed withdrawals and operational disruption; no direct loss.

RECOMMENDATION

Revert if the first deleted output is already finalised.

FIX

contracts/L1/OutputOracle.sol (remediation commit)	DIFF
1 + require(block.timestamp - l2Outputs[_l2OutputIndex].timestamp < finalizationPeriodSeconds , "OutputOracle: output finalized");	

RESOLUTION

Fixed as recommended.

REFERENCES

- CWE-284: Improper Access Control
<https://cwe.mitre.org/data/definitions/284.html>
- OpenZeppelin Contracts: Access Control
<https://docs.openzeppelin.com/contracts/5.x/access-control>

OR-08 ■ Low Resolved

Batcher logs its configuration, including the signer key path and RPC credentials, at debug level

op-sequencer/batcher/config.go:L96

Category	CWE	Likelihood	Impact
Key management	CWE-532	Low	Medium

DESCRIPTION

On start-up the batcher prints its full configuration struct with `log.Debug("config", "cfg", cfg)`. The struct contains the L1 RPC URL with an embedded API token and, when a local key is used instead of a remote signer, the private key in hex. Debug logging is enabled in the staging environment and logs are shipped to a shared aggregation service.

PRECONDITIONS

- Debug logging enabled and access to the log service.

EXPLOIT SCENARIO

1. An engineer with log access, or an attacker who compromises the log service, reads the batcher key and posts batches as the batcher.

IMPACT

Compromise of the batcher key allows posting arbitrary batch data, disrupting derivation until the key is rotated in SystemConfig.

RECOMMENDATION

Implement redaction for secret fields and use a remote signer in every environment.

FIX

op-sequencer/batcher/config.go (remediation commit)	DIFF
1 - log.Debug("config", "cfg", cfg) 2 + log.Debug("config", "cfg", cfg.Redacted()) // PrivateKey, L1RPCURL credentials replaced with "[redacted]"	

RESOLUTION

Fixed. Secrets are redacted and staging now uses the remote signer.

REFERENCES

- CWE-532: Insertion of Sensitive Information into Log File
<https://cwe.mitre.org/data/definitions/532.html>

OR-09 ■ Informational Acknowledged

SystemConfig setters do not emit dedicated events

contracts/L1/SystemConfig.sol (setGasLimit, setResourceConfig)

Category	CWE	Likelihood	Impact
Code quality	—	Low	Low

DESCRIPTION

Configuration changes emit a generic `ConfigUpdate(version, type, data)` event with ABI-encoded data. Monitoring has to decode the payload to distinguish a gas-limit change from a batcher rotation, which slows alerting on the most sensitive change (batcher key rotation).

PRECONDITIONS

- None; observability only.

EXPLOIT SCENARIO

1. Not exploitable.

IMPACT

Slower detection of configuration changes.

RECOMMENDATION

Keep the generic event for compatibility and add typed events for batcher and resource-configuration changes.

RESOLUTION

Acknowledged. Obsidian keeps the generic event for tooling compatibility and has added decoding rules to its monitoring.

REFERENCES

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

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	1 OR-02	1 OR-03	1 OR-01
Medium	2 OR-07 OR-08	3 OR-04 OR-05 OR-06	0
Low	1 OR-09	0	0
	Low	Medium	High

Likelihood

Distribution by vulnerability class

Denial of service	3
Access control	1
Business logic	1
Code quality	1
Governance manipulation	1
Key management	1
Unsafe proxy upgrade	1

Severity distribution



9

Remediation status

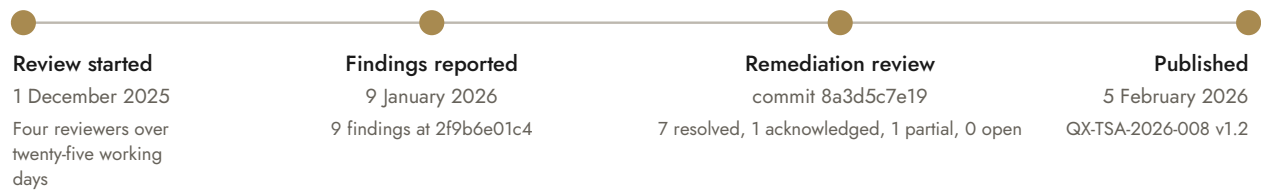
7 of 9 findings are resolved, 1 acknowledged, 1 partially resolved and 0 open, as reviewed at remediation commit 8a3d5c7e19f0.

Severity	Resolved	Partially resolved	Acknowledged	Open	Total
■ Critical	1	0	0	0	1
■ High	1	0	0	0	1
■ Medium	3	1	0	0	4
■ Low	2	0	0	0	2
■ Informational	0	0	1	0	1

Remediation progress

94%

Remediation history



Residual items

ID	Status	Notes
OR-02	Partially resolved	Partially resolved. Routine upgrades now pass through a 48-hour <code>UpgradeTimeLock</code> , shorter than the eight days recommended, so users still cannot always exit before an upgrade takes effect; the 4-of-7 security council retains an instant path for emergencies, which Obsidian documents in its public risk disclosures.
OR-09	Acknowledged	Acknowledged. Obsidian keeps the generic event for tooling compatibility and has added decoding rules to its monitoring.

10

Formal verification and invariants

The Portal withdrawal state machine and the dispute-game bisection were specified as properties and checked with bounded symbolic execution; storage layouts were compared mechanically. Finalisation-safety properties failed on the original commit (OR-03, OR-04) and hold on the remediation commit. A global property relating output finalisation to in-progress games was not completed.

Tool Halmos and Foundry symbolic tests; storage layouts compared with forge inspect

ID	Property	Result	Notes
P1	A withdrawal hash is finalised at most once, regardless of the target call result.	■ Proved	Violated on the original commit when the target call failed (OR-04).
P2	Finalisation requires a dispute game resolved as DEFENDER_WINS for at least the finality delay.	■ Proved	Violated on the original commit (OR-03).
P3	A bisection game terminates within MAX_GAME_DEPTH moves on each branch.	■ Proved	—
P4	The OutputOracle v2 storage layout is an append-only extension of v1.	■ Proved	Checked mechanically on the remediation commit; failed on the original v2 (OR-01).
P5	No output root becomes final while a game disputing it is in progress.	■ Inconclusive	Requires reasoning across contracts and time; covered by differential tests instead.

11

Recommendations

Gate every upgrade on a storage-layout diff

Make the layout comparison a required CI step that produces the upgrade calldata, so a failing check cannot be bypassed manually.

Publish the upgrade and emergency powers

Document the timelock delay, the security council threshold and the conditions for emergency action alongside the rollup's risk disclosures.

Run a permanent honest challenger

Operate challenger agents from independent infrastructure and fund their bonds so that fault proofs remain effective.

Monitor game lifecycle and withdrawals

Alert on games approaching resolution, on blacklisting, and on finalisations of large withdrawals.

Adopt a remote signer everywhere

Remove local private keys from batcher and proposer configuration in all environments.

12

Limitations

- The L2 execution client, derivation pipeline and fault-proof VM were out of scope; their correctness is assumed.
- The review covered the v2 upgrade as scripted at the stated commit; later changes to the script are not covered.
- Reorg handling was tested on a local devnet with simulated reorganisations up to depth 4.
- Bounded symbolic results hold up to the stated depth and game size.

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, differential and fork tests
Halmos	Bounded symbolic tests of the Portal and dispute game
forge inspect storageLayout	Storage-layout comparison between implementations
Slither	Static analysis of Solidity
staticcheck and golanci-lint	Static analysis of Go
Local devnet	L1 reorganisation simulation

A.2 Reviewed files (SHA-256)

File	SHA-256
contracts/L1/Portal.sol	2baceed1ad4e8858b07790c1428ee9ce160d9234583a8470babf12ea63b240d2
contracts/L1/OutputOracle.sol	3f1321a65c1c7ebf481b6cc66a6df1317bcfc553bede7b2f0b832bba519d03ab
contracts/dispute/DisputeGame.sol	98284271b049a3fd72e9b2a751a14c9a5fb210e963cfd9ba0613888ff976af7a
contracts/L1/BatchInbox.sol	ceccf63e29779736d472be8070175231a1a821604a64d75d0ccdd3b78102434b
contracts/L1/SystemConfig.sol	f27cef5c452c607ff0d95275d113afe86b17d7afe8b53c7cbbe4c675e36e37d5
op-sequencer/batcher/tx_manager.go	e87f950cd83a43ea69baabbce31cdebcc91f68efa5048424d5619af5483b8498

A.3 Notes

- File hashes are SHA-256 digests of the reviewed sources at the audited commit.
- OR-01 concerned an upgrade that had not been executed; no deployed contract was affected.

A.4 Methodology and technical references

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

1. EIP-1967: Proxy storage slots
<https://eips.ethereum.org/EIPS/eip-1967>
2. EIP-7201: Namespaced storage layout
<https://eips.ethereum.org/EIPS/eip-7201>
3. OpenZeppelin: Writing upgradeable contracts
<https://docs.openzeppelin.com/updates-plugins/writing-upgradeable>
4. OP Stack specifications
<https://specs.optimism.io/>
5. CWE-841: Improper Enforcement of Behavioral Workflow
<https://cwe.mitre.org/data/definitions/841.html>
6. CWE-754: Improper Check for Unusual or Exceptional Conditions
<https://cwe.mitre.org/data/definitions/754.html>

7. EIP-1559: Fee market change
<https://eips.ethereum.org/EIPS/eip-1559>
8. CWE-284: Improper Access Control
<https://cwe.mitre.org/data/definitions/284.html>
9. OpenZeppelin Contracts: Access Control
<https://docs.openzeppelin.com/contracts/5.x/access-control>
10. CWE-532: Insertion of Sensitive Information into Log File
<https://cwe.mitre.org/data/definitions/532.html>
11. Solidity documentation: Events
<https://docs.soliditylang.org/en/latest/contracts.html#events>

A.5 Document information

Document	Quantix Technical Security Assessment QX-TSA-2026-008, version 1.2
Subject	Obsidian Rollup, Quantix Protocol Research
Status	Published 5 February 2026
Disclosure	This document is a Quantix research assessment of Obsidian Rollup, 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.