ESPRESSO SYSTEMS

# Optimism-Espresso Integration

Internal Security Audit: OP Streamer & TEE Contracts

**Team:**
Internal Security Team
2026-01-29
v1.0

**Scope:**
OP Streamer Component
TEE Contracts
**security@espressosys.com**

## Table of Contents

# 1 Executive Summary

This report presents the findings of a security audit of the Optimism-Espresso integration, focusing on the OP Streamer component and TEE (Trusted Execution Environment) contracts. The audit identified **14 vulnerabilities** (2 Critical, 4 High, 1 Medium, 7 Low) across batch streaming logic, TEE networking, and smart contract implementations.

**Key findings from this assessment include 2 Critical, 4 High, 1 Medium, and 7 Low vulnerabilities identified.** The critical and several high-severity issues have been addressed in PR #43 of the espresso-tee-contracts repository. The remaining findings require attention to strengthen the overall security posture of the integration.

We recommend addressing the identified findings through the standard development cycle and implementing the security recommendations outlined in the detailed findings section to further strengthen the overall security posture of the integrated platform.

## 1.1 Severity Distribution

| Severity | Count | Fixed | Open |
|---|---|---|---|
| Critical | 2 | 2 | 0 |
| High | 4 | 1 | 3 |
| Medium | 1 | 0 | 1 |
| Low | 7 | 0 | 7 |
| **Total** | **14** | **3** | **11** |

## 1.2 Key Findings Summary

| ID | Vulnerability | Severity | Component | Status |
|---|---|---|---|---|
| V-4 | Cross-Chain Deployment | Critical | TEE Contracts | Fixed |
| V-6 | Missing Journal Validations | Critical | TEE Contracts | Fixed |
| V-2 | Infinite Buffer Growth | High | OP Streamer | Open |
| V-3 | TEE Networking MitM | High | TEE Enclave | Open |
| V-5 | Signer Deletion DoS | High | TEE Contracts | Fixed |
| V-7 | Type Mismatch in Refresh() | High | OP Streamer | Open |
| V-1 | All-At-Once RPC Calls | Medium | OP Streamer | Open |
| V-8 | Missing Duplicate Detection | Low | OP Streamer | Open |
| V-9 | Misleading Log Messages | Low | OP Streamer | Open |
| V-10 | Inefficient Batch Overwrite | Low | OP Streamer | Open |

| ID | Vulnerability | Severity | Component | Status |
|---|---|---|---|---|
| V-11 | Confusing Variable Naming | Low | OP Streamer | Open |
| V-12 | Unused Constant Declaration | Low | OP Streamer | Open |
| V-13 | Missing Sort Order Validation | Low | OP Streamer | Open |
| V-14 | No Network Failure Distinction | Low | OP Streamer | Open |

# 2 Scope and Methodology

## 2.1 Audit Scope

This audit covers the following components:

- **OP Streamer Component** (`espresso/`)
  - `batch_buffer.go` – Batch buffering and ordering logic
  - `streamer.go` – Espresso block streaming and batch processing
  - `buffered_streamer.go` – Buffered streaming implementation
- **TEE Contracts** (`espresso-tee-contracts`)
  - `EspressoNitroTEEVerifier.sol` – AWS Nitro attestation verification
  - `EspressoSGXTEEVerifier.sol` – Intel SGX attestation verification
  - `TEEHelper.sol` – Shared TEE helper functionality
  - Related interfaces and libraries

## 2.2 Methodology

- **Static Code Analysis**: Manual review of Go and Solidity source code
- **Architecture Review**: Analysis of component interactions and data flow
- **Attack Scenario Modeling**: Threat modeling for potential exploits
- **Documentation Review**: Analysis of security documentation and deployment guides

# 3 OP Streamer Component Vulnerabilities

## 3.1 V-1: All-At-Once RPC Calls

**Criticality:** Medium
**Status:** Open
**Component:** `espresso/streamer.go` – `CheckBatch()` and `processRemainingBatches()`

### 3.1.1 Overview

The `CheckBatch` function makes synchronous L1 RPC calls (`HeaderHashByNumber`) when validating finalized batches. When multiple batches become finalized simultaneously, the system executes sequential synchronous RPC calls, causing the streamer to freeze.

### 3.1.2 Description

**Vulnerable Code Path:**

```
func CheckBatch(batch B, l1Origin eth.BlockID) {
    if isFinalized(l1Origin) {
        hash := HeaderHashByNumber(l1Origin.Number) // Synchronous RPC
        // ... validation logic
    }
}
```

**Attack Scenario:**

1. Node accumulates 500 batches in `RemainingBatches` while waiting for L1 finality
2. L1 finalizes a new state
3. `processRemainingBatches()` iterates all 500 batches
4. Finalized check now passes for all batches
5. System executes **500 sequential synchronous RPC calls** inside the `Update` loop

### 3.1.3 Impact

- **Availability**: Streamer freezes for seconds to minutes
- **Denial of Service**: Node stops fetching new Espresso blocks
- **Cascading Failure**: Downstream components dependent on streamer become blocked

### 3.1.4 Recommendation

1. **Immediate**: Implement batch RPC calls using `eth_getBlockByNumber` with multicall
2. **Short-term**: Add asynchronous RPC call handling with worker pool

3. **Long-term**: Cache L1 block hashes and implement rate limiting

## 3.2 V-2: Infinite Buffer Growth

**Criticality:** High
**Status:** Open
**Component:** `espresso/batch_buffer.go` – `BatchBuffer` and `HasNext()`

### 3.2.1 Overview

The `BatchBuffer` has no size limit and will accept batches indefinitely while waiting for a missing batch, leading to memory exhaustion and node crashes.

### 3.2.2 Description

**Vulnerable Logic:**

```
func (b *BatchBuffer[B]) HasNext() bool {
    return b.Peek() == b.expectedBatchPos
}
```

**Attack Scenario:**

1. Node expects Batch #100
2. Espresso network delivers Batch #101, #102, ... #50,000
3. Batch #100 is missing (network partition, Byzantine node, etc.)
4. `BatchBuffer` accepts and stores Batches #101 through #50,000 in memory
5. Node waits indefinitely for Batch #100
6. Memory exhaustion leads to node crash

### 3.2.3 Impact

- **Availability**: Node crashes due to out-of-memory (OOM)
- **Denial of Service**: Missing batch prevents all downstream processing
- **No Recovery**: No mechanism to invalidate the stream and skip missing batch

### 3.2.4 Recommendation

1. **Critical**: Implement maximum buffer size (e.g., 1000 batches)
2. **Critical**: Add timeout for missing batches (e.g., 10 minutes)
3. **Important**: Implement gap detection and alerting
4. **Important**: Add mechanism to request missing batches from peers
5. **Long-term**: Implement stream reset when gap is detected beyond threshold

**Suggested Implementation:**

```
const MAX_BUFFER_SIZE = 1000
const MISSING_BATCH_TIMEOUT = 10 * time.Minute

func (b *BatchBuffer[B]) TryInsert(batch B) (int, bool) {
    if len(b.batches) >= MAX_BUFFER_SIZE {
        return 0, false  // Reject new batches
    }

    if batch.Number > b.expectedBatchPos {
        if time.Since(b.lastProgressTime) > MISSING_BATCH_TIMEOUT {
            // Log critical error and reset stream
            return 0, false
        }
    }

    // ... existing logic
}
```

## 3.3 V-7: Type Mismatch in Refresh()

**Criticality:** High
**Status:** Open
**Component:** `espresso/streamer.go` – `Refresh()` function, Line 173

### 3.3.1 Overview

The `Refresh()` function contains a type mismatch where it compares `fallbackBatchPos` (representing a Batch Index) with `hotShotPos` (representing an Espresso Block Height). These are incompatible types that should not be directly compared.

### 3.3.2 Description

**Vulnerable Code:**

```
func (s *BatchStreamer[B]) Refresh(ctx context.Context,
    finalizedL1 eth.L1BlockRef, safeBatchNumber uint64,
    safeL1Origin eth.BlockID) error {
    // Line 173
    if fallbackBatchPos < hotShotPos {  // Type mismatch!
        // ... logic
    }
}
```

**Issue:**

- `fallbackBatchPos` is a **Batch Index** (sequential batch number)
- `hotShotPos` is an **Espresso Block Height** (blockchain height)
- Comparing these directly may lead to logic errors

### 3.3.3 Impact

- **State Inconsistency**: Incorrect state transitions during batch processing
- **Logic Error**: May cause unexpected behavior in edge cases
- **Potential Data Loss**: Could skip or process wrong batches

### 3.3.4 Recommendation

1. Review the comparison logic and ensure type compatibility
2. Add type-safe wrappers or explicit conversions
3. Document the relationship between batch index and block height
4. Add assertions to validate the comparison is meaningful

## 3.4 V-8 through V-14: Additional Low Severity Issues

### 3.4.1 V-8: Missing Duplicate Detection

**Criticality:** Low **Status:** Open **Component:** `batch_buffer.go`

The `Insert(batch B, i int)` function unconditionally inserts a batch without checking for duplicates, potentially causing data redundancy and memory waste.

### 3.4.2 V-9: Misleading Log Messages

**Criticality:** Low **Status:** Open **Component:** `streamer.go`

Lines 304 and 435 contain misleading and redundant log messages that make debugging more difficult and could confuse operators.

### 3.4.3 V-10: Inefficient Batch Overwrite

**Criticality:** Low **Status:** Open **Component:** `streamer.go`

Line 435 unnecessarily overwrites a batch in the `RemainingBatches` map when the batch already exists, performing redundant work.

### 3.4.4 V-11: Confusing Variable Naming

**Criticality:** Low **Status:** Open **Component:** `cli.go`

The configuration variable `PollingHotShotPollingInterval` contains redundant naming that reduces code readability.

### 3.4.5 V-12: Unused Constant Declaration

**Criticality:** Low **Status:** Open **Component:** `espresso/`

The constant `HOTSHOT_BLOCK_STREAM_LIMIT` is defined but never used, leading to dead code.

### 3.4.6 V-13: Missing Sort Order Validation

**Criticality:** Low **Status:** Open **Component:** `batch_buffer.go`

The `TryInsert()` function assumes the batch list is sorted and uses binary search without verifying this invariant.

### 3.4.7 V-14: No Network Failure Distinction

**Criticality:** Low **Status:** Open **Component:** `streamer.go`

The `confirmEspressoBlockHeight()` function returns `false` when RPC calls fail, making it impossible to distinguish between actual state verification and network errors.

# 4 TEE Enclave Vulnerabilities

## 4.1 V-3: TEE Networking MitM Attack

**Criticality:** High
**Status:** Open
**Component:** TEE Enclave Networking Layer

### 4.1.1 Overview

The TEE enclave networking layer lacks sufficient protection against Man-in-the-Middle (MitM) attacks, potentially allowing malicious actors to feed the enclave with arbitrary input, such as maliciously crafted HotShot blocks.

### 4.1.2 Description

**Vulnerability:**

- TEE networking layer does not enforce strict TLS certificate validation
- No certificate pinning implemented
- Enclave trusts any valid TLS certificate

**Attack Scenario:**

1. Attacker positions themselves between TEE enclave and HotShot network
2. Attacker presents valid TLS certificate (e.g., from compromised CA)
3. TEE accepts connection as legitimate
4. Attacker injects malicious HotShot blocks
5. TEE processes fraudulent data as authentic

### 4.1.3 Impact

- **Integrity**: TEE processes malicious input as authentic
- **Consensus Manipulation**: Fraudulent blocks could affect L2 state
- **Trust Violation**: Undermines security guarantees of TEE

### 4.1.4 Current Mitigation

Documentation exists at: https://eng-wiki.espressosys.com/mainch36.html

However, implementation remains vulnerable.

#### 4.1.5 Recommendation

1. **Critical**: Implement certificate pinning
    - Embed expected certificates during enclave build
    - Include certificates in PCR0 hash measurement
    - Ensure enclave only trusts specific, validated endpoints
2. **Important**: Add certificate rotation mechanism
    - Design automatic certificate update process
    - Implement gradual rollover to avoid service interruption
3. **Long-term**: Implement attestation-based mutual authentication
    - Both endpoints verify each other's TEE attestations
    - Remove dependency on traditional PKI

# 5 TEE Contracts Vulnerabilities

The following vulnerabilities were identified and **fixed** in PR #43 of the `espresso-tee-contracts` repository.

**PR #43 Summary**

**Title:** Internal Audit #2 – Security Fixes
**Merged:** January 28, 2026
**Commit:** `1a5a179`
**Files Changed:** 21 files (+1098, -250 lines)
**Test Coverage:** 624 new test lines added

## 5.1 V-4: Cross-Chain Deployment Vulnerability

**Criticality:** Critical
**Status:** Fixed
**Component:** `EspressoNitroTEEVerifier.sol`, `EspressoSGXTEEVerifier.sol`
**Fix Reference:** Commit `57bf5de`, PR #43

### 5.1.1 Overview

TEE Verifier contracts maintain chain-specific on-chain state for registered enclaves and signers. However, attestations are not chain-specific by default, allowing replay attacks across different chains with inconsistent security policies.

### 5.1.2 Description

**Vulnerable State Management:**

```
// These mappings are stored ON-CHAIN (chain-specific):
mapping(ServiceType => mapping(bytes32 => bool))
    public registeredEnclaveHashes;
mapping(ServiceType => mapping(address => bool))
    public registeredServices;
```

**Problem:** State is local to each chain, but attestations can be replayed across chains.

### 5.1.3 Attack Scenarios

**Attack 1: Uncoordinated Revocation**

- Day 1: Enclave hash approved on Ethereum and Arbitrum
- Day 30: Vulnerability discovered in enclave
- Day 31: Hash revoked on Ethereum
- **Result:** Attacker blocked on Ethereum [OK] but still valid on Arbitrum [FAIL]

**Attack 2: Attestation Replay**

1. TEE generates single attestation
2. Attacker registers on Ethereum using attestation
3. Attacker reuses **same attestation** on Arbitrum
4. Attacker reuses **same attestation** on Optimism
5. All registrations succeed (if hash is approved on each chain)

**Attack 3: Policy Inconsistency**

- Ethereum: High security, only approves hash v2.0 (latest, secure)
- Arbitrum: Different governance, approves hash v1.0 (old, vulnerable)
- **Result:** Same codebase, different security across chains

### 5.1.4 Impact

- **Security Fragmentation**: Inconsistent security policies across chains
- **Delayed Response**: Vulnerability on one chain doesn't automatically propagate
- **Replay Attacks**: Single attestation usable on multiple chains

### 5.1.5 Fix Applied

Added Security Considerations section in README.md documenting multi-chain deployment risks and best practices.

## 5.2 V-5: Signer Deletion DoS Attack

**Criticality:** High
**Status:** Fixed
**Component:** `TEEHelper.sol`
**Fix Reference:** Commit `3026966`, PR #43

### 5.2.1 Overview

The TEE Helper contract iterates over a list of registered signers in deletion operations. An attacker could exploit unbounded loops to cause denial of service by exceeding block gas limits.

### 5.2.2 Description

**Attack Scenario:**

1. Attacker registers many signers (e.g., 10,000 addresses)
2. Enclave is compromised
3. Operator tries to revoke by calling `setEnclaveHash(hash, false)` and `deleteRegisteredSigners()`
4. **Deletion fails due to gas limit** – transaction reverts
5. **Compromised signers remain active** – security breach!

### 5.2.3 Impact

- **Security Bypass**: Unable to fully revoke compromised enclave access
- **DoS on Critical Security Function**: Deletion operation required but impossible
- **Persistent Vulnerability**: Compromised signers remain valid indefinitely

### 5.2.4 Fix Applied in PR #43

Changed the security model so that **deleting signers is no longer required**. Revoking an enclave hash via `setEnclaveHash(hash, false)` is now sufficient to prevent new malicious registrations.

| Signers | Gas Cost | Block Limit (30M) | Status |
|---|---|---|---|
| 100 | ∼500k | [OK] Safe | OK |
| 1,000 | ∼5M | [OK] Safe | OK |
| 5,000 | ∼25M | [WARN] Close | Risk |
| 10,000 | ∼50M | [FAIL] Over | DoS |

**Why This Works:**

- When an enclave is compromised, the private keys are already exposed to attackers

- Existing signer addresses in the registry don’t grant any additional attack surface
- The security boundary is enforced at enclave hash validation, not signer presence
- Revoking the hash immediately protects the system

## 5.3 V-6: Missing TEE Journal Struct Validations

**Criticality:** Critical
**Status:** Fixed
**Component:** `EspressoNitroTEEVerifier.sol`
**Fix Reference:** Commit `c47d9aa`, PR #43

### 5.3.1 Overview

The VerifierJournal struct contains critical cryptographic fields (PCRs, public key, nonce, timestamp, userData) that require comprehensive validation. Missing validations could allow malformed attestations to be accepted, potentially leading to predictable signer addresses or other cryptographic attacks.

### 5.3.2 Description

**Journal Structure:**

```
struct VerifierJournal {
    bytes32[] pcrs;          // Platform Configuration Registers
    bytes publicKey;         // Enclave public key (should be 65 bytes)
    bytes nonce;             // Replay protection
    uint256 timestamp;       // Attestation time
    bytes userData;          // Application data
    string moduleId;         // Nitro module identifier
    VerificationResult result;
}
```

### 5.3.3 Specific Vulnerabilities

**V-6a: Empty PCR Array**

- **Issue**: No validation that PCR array contains data
- **Impact**: Could accept attestations without platform measurements
- **Exploit**: Bypass hardware attestation requirements

**V-6b: Invalid Public Key Format**

- **Issue**: No check for correct public key length (65 bytes) and format
- **Impact**: Malformed public keys could lead to predictable addresses

**V-6c: Predictable Signer Addresses**

- **Issue**: Invalid public key formats can produce predictable Ethereum addresses
- **Impact**: Attacker could precompute and claim desirable addresses
- **Severity**: Enables address squatting and impersonation

### 5.3.4 Fix Applied

Added comprehensive `_validateJournal()` validation function:

```
function _validateJournal(VerifierJournal memory journal)
    internal view {
    // 1. Validate PCR array bounds
    require(journal.pcrs.length > 0, "PCR array cannot be empty");

    // 2. CRITICAL: Validate public key format
    require(journal.publicKey.length == 65,
        "Invalid public key length");
    require(journal.publicKey[0] == 0x04,
        "Public key must be uncompressed");

    // 3. Note: Nonce validation removed - AWS Nitro may have
    //    empty nonce. Implement nonce tracking separately
    //    if replay protection needed

    // 4. Timestamp validation already done by NitroEnclaveVerifier
    //    Result would be InvalidTimestamp if timestamp is bad
}
```

# 6 Conclusion

This security audit identified 14 vulnerabilities across the Optimism-Espresso integration codebase. While 3 critical and high-severity issues have been successfully addressed in PR #43, 11 vulnerabilities remain open and require attention.

## 6.1 Priority Recommendations

1. **Immediate Action Required:**
    - V-2: Implement buffer size limits to prevent memory exhaustion
    - V-3: Implement certificate pinning for TEE networking
    - V-7: Fix type mismatch in Refresh() function
2. **Short-term Improvements:**
    - V-1: Optimize RPC call batching
    - Low-severity code quality issues (V-8 through V-14)
3. **Long-term Monitoring:**
    - Continue monitoring fixed vulnerabilities
    - Implement comprehensive testing for edge cases
    - Regular security audits of new features

# 7 List of Changes

| Version | Date | Description | Author |
|---|---|---|---|
| 0.1 | 2026-01-27 | Initial draft with findings | Internal Security Team |
| 0.5 | 2026-01-28 | Added PR #43 fix details | Internal Security Team |
| 1.0 | 2026-01-29 | Final report | Internal Security Team |

# 8 Disclaimer

This report is a comprehensive review of in-scope items and should not be considered an endorsement or disapproval of any facet of the Espresso Systems software platform. It does not guarantee or provide any warranty regarding the absolute continuity of the analyzed technologies. Nor does it reflect the economics, value, business model, or legal compliance of such products. This report should not be used to make any investment decisions, nor does it constitute investment advice. This report serves as a comprehensive evaluation of the technology and constituent components, aimed at improving the overall security posture of the platform.

# 9 Severity Measurement Scale

Espresso Systems measures severity using in-house expertise along with industry standard methodologies as outlined in the Open Web Application Security Project (OWASP) and the Common Vulnerability Scoring System (CVSS).

The severity of each finding in this report is established after the conclusion of rigorous testing, in accordance with the threat model outlined in the statement of work. We evaluate the severity of findings in isolation, irrespective of other findings. A vulnerability's overall business and technical impact is correlated to its score, where a higher score corresponds to higher business and technological impact, and lower scores are less severe.

c

| | |
|---|---|
| **Critical** | Vulnerabilities that score "high" but additionally pose a severe business or security impact. Examp |
| **High** | Vulnerabilities that pose a significant risk to the business, such as unauthorized access to sens |
| **Medium** | Vulnerabilities that do not cause immediate harm but may be used to escalate privileges |
| **Low** | Vulnerabilities with minimal security impact that are unlikely to be exploited but may still impr |

# 10 Imprint

Espresso Systems
Internal Security Team
security@espressosys.com

**Last Updated:** January 29, 2026
**Version:** 1.0