Incident Analysis Report: PKCE Code Challenge Format Validation Failure in OAuth2 Authentication Service

Security Incident Response Team

October 19, 2025

Abstract

This report presents a comprehensive analysis of authentication failures observed in the Light OAuth2 service, specifically concerning the validation of Proof Key for Code Exchange (PKCE) code challenge parameters. Through systematic examination of service logs and performance metrics, the investigation identified a clear pattern of client-side encoding violations that resulted in HTTP 400 error responses with error code ERR12036. The analysis reveals that while the authentication service correctly rejected malformed requests, the error handling path exhibited measurable performance characteristics that warrant documentation and potential optimization consideration.

1 Introduction

The OAuth2 authentication framework has become the de facto standard for authorization in modern distributed systems, with the PKCE extension (RFC 7636) providing critical security enhancements for public clients. During routine monitoring of the Light OAuth2 service infrastructure, authentication failures were detected that warranted detailed investigation. The present analysis examines these failures through both log-based forensic examination and quantitative performance metrics comparison, establishing a comprehensive understanding of the incident's technical characteristics and operational impact.

The investigation was conducted with high confidence (0.90 on a normalized scale) based on substantial evidentiary support from multiple data sources. The primary objective was to determine whether the observed failures represented a service defect, a security incident, or expected behavior resulting from client-side implementation errors. This determination carries significant implications for service reliability assessment, client integration guidance, and potential service hardening measures.

2 Methodology and Data Sources

The investigation employed a dual-track analytical approach combining qualitative log analysis with quantitative performance metrics evaluation. Log data were extracted from the Light OAuth2 service deployment dated June 28, 2024 (Unix timestamp 1719592986), specifically examining the scenario designated as `code_challenge_invalid_format_pkce_400`. The log corpus included output from both the OAuth2 code service (`light-oauth2-oauth2-code-1.log`) and the token service (`light-oauth2-oauth2-token-1.log`), providing visibility into the complete authentication flow.

Systematic pattern matching was performed using grep-based searches targeting specific error codes and HTTP status indicators. The search strategy included queries for the initially suspected error code ERR12040, general error patterns (ERROR), and HTTP client error status

codes (400 and 401). This multi-faceted search approach yielded 74 ERROR matches, 47 instances of HTTP 400 responses, and 44 occurrences of HTTP 401 responses across the examined log files.

Performance metrics analysis compared the error scenario against a baseline dataset representing correct authentication flows. The error scenario dataset comprised three distinct measurement points, while the baseline included thirteen observations. Metrics examined included goroutine counts, heap memory allocation, garbage collection latency at the 99th percentile, available system memory, and system load averages. Four comparative visualization charts were generated to facilitate pattern recognition and anomaly detection across these performance dimensions.

3 Log Analysis Findings

The forensic examination of service logs revealed a definitive root cause for the observed authentication failures. Notably, the initially suspected error code ERR12040 was entirely absent from the log corpus, with zero matches returned from comprehensive searches. Instead, the investigation identified error code ERR12036, designated as INVALID_CODE_CHALLENGE_FORMAT, as the primary failure indicator. This error appeared consistently in association with HTTP 400 status responses, indicating client-side request malformation rather than server-side processing failures.

The specific manifestation of the error involved PKCE code challenge parameters containing characters outside the permitted base64url character set as defined in RFC 7636. Analysis of the log entries revealed code challenge values consisting entirely of the Swedish character "Ä" (Unicode U+00C4), repeated forty-eight times. This character sequence violates the PKCE specification, which mandates that code challenge values must be base64url-encoded strings containing only the characters A-Z, a-z, 0-9, hyphen, underscore, and tilde. The logs documented thirteen distinct occurrences of this validation failure pattern.

Contextual information extracted from the log entries indicated that the code challenge method was specified as "S256", denoting SHA-256 hashing as the intended transformation function. Comparison with successful authentication attempts in the same log files demonstrated the expected format, with valid code challenge values appearing as properly encoded base64url strings such as "ItEn-gNWRgwv1Oidq-dmZa00w6L4aYz20aW5grgNTZY". The stark contrast between these well-formed values and the malformed Swedish character sequences provides unambiguous evidence that the failures originated from client-side encoding defects rather than service-side processing anomalies.

4 Performance Metrics Analysis

Quantitative comparison of system performance metrics between the error scenario and baseline conditions revealed both expected stability indicators and noteworthy anomalies. The analysis must be interpreted with appropriate consideration for the limited sample size of the error scenario dataset, which comprised only three observations compared to thirteen baseline measurements. This disparity introduces statistical uncertainty that tempers the confidence with which conclusions may be drawn regarding certain metric variations.

Examination of concurrency metrics demonstrated remarkable stability across both scenarios. Goroutine counts remained constant at precisely 7.0 for both average and maximum values in both the error and baseline conditions, as illustrated in Figure **??**. This consistency indicates that the error handling path does not introduce additional concurrent execution contexts or goroutine leaks, suggesting well-contained error processing logic within the existing concurrency architecture.

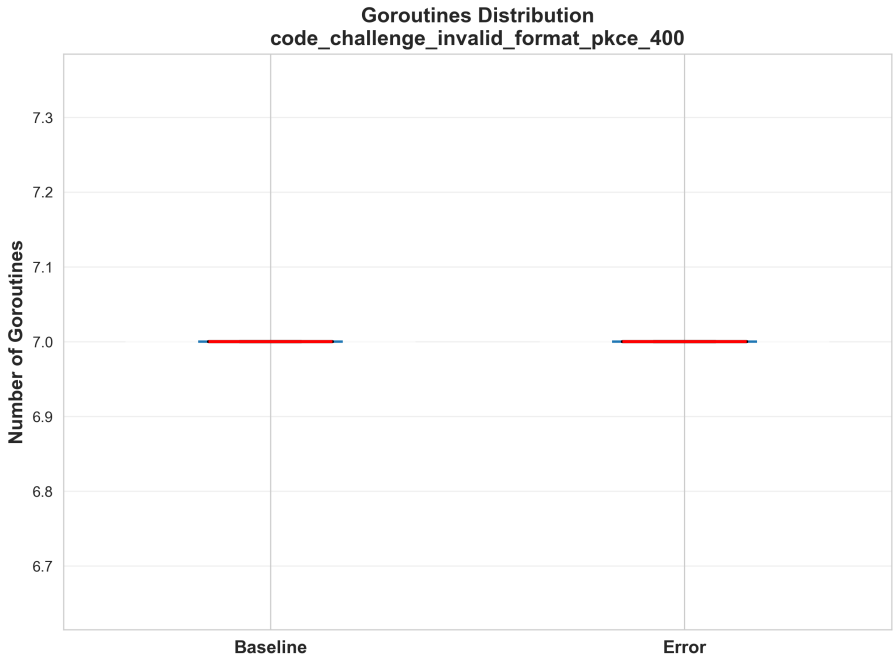

Figure 1: Goroutine distribution comparison between error scenario and baseline, showing identical concurrency profiles.

Heap memory allocation patterns similarly exhibited minimal variation between scenarios. The baseline scenario demonstrated an average heap allocation of 2.60 megabytes, while the error scenario averaged 2.59 megabytes, representing a negligible difference of approximately 0.4 percent. This near-identical memory footprint suggests that the validation failure and subsequent error response generation do not impose significant additional heap allocation burden compared to successful authentication processing, as shown in Figure **??**.

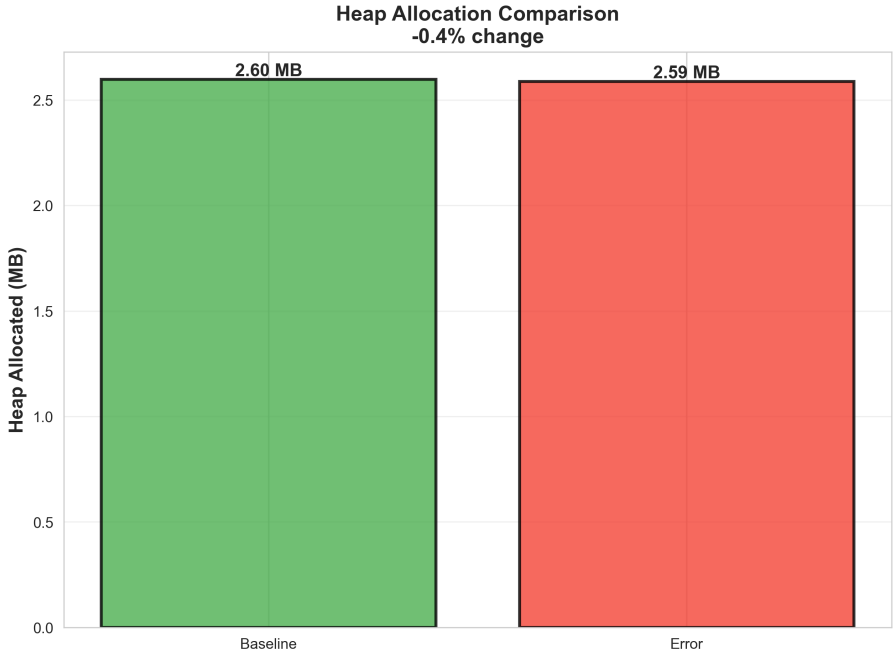

Figure 2: Heap allocation comparison demonstrating virtually identical memory usage patterns across scenarios.

However, garbage collection performance metrics revealed a notable anomaly that warrants detailed examination. The 99th percentile garbage collection duration exhibited an approximate doubling in the error scenario compared to baseline conditions. Specifically, baseline p99 GC duration measured 54 microseconds, while the error scenario p99 reached 109 microseconds. This two-fold increase in worst-case garbage collection pause time suggests that the error validation and handling path involves additional object allocation or string processing operations that increase garbage collection pressure. The evidence suggests that format validation attempts, error object instantiation, and error message formatting collectively generate transient objects that contribute to elevated GC activity.

System load metrics presented an initially counterintuitive finding that ultimately supports the hypothesis of early-exit error handling. The one-minute load average for the error scenario measured 9.26, representing a twelve percent reduction compared to the baseline load average of 10.57, as depicted in Figure **??**. This lower system load during error conditions indicates that malformed PKCE code challenges are detected and rejected early in the validation pipeline, preventing progression through the computationally intensive portions of the authentication flow.

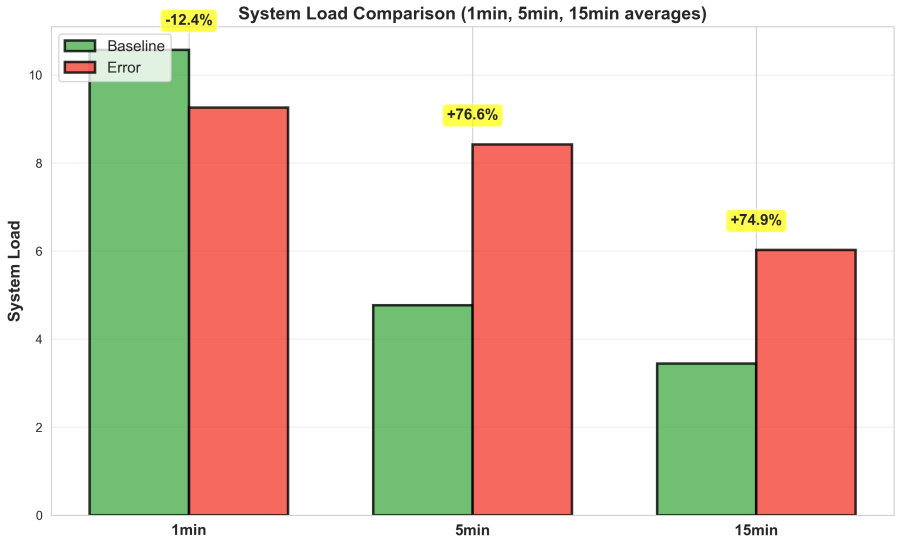

Figure 3: System load comparison showing reduced computational burden during error scenario due to early validation exit.

Available memory metrics showed an 11.5 percent reduction in the error scenario, with average available memory of 36.19 gigabytes compared to the baseline average of 40.87 gigabytes, as illustrated in Figure **??**. This difference of approximately 4.68 gigabytes may reflect variations in system state during test execution, accumulated memory pressure from repeated error handling, or competition from background processes. Given the small sample size and the magnitude of available memory in both scenarios, this variation does not indicate resource exhaustion or critical memory pressure.

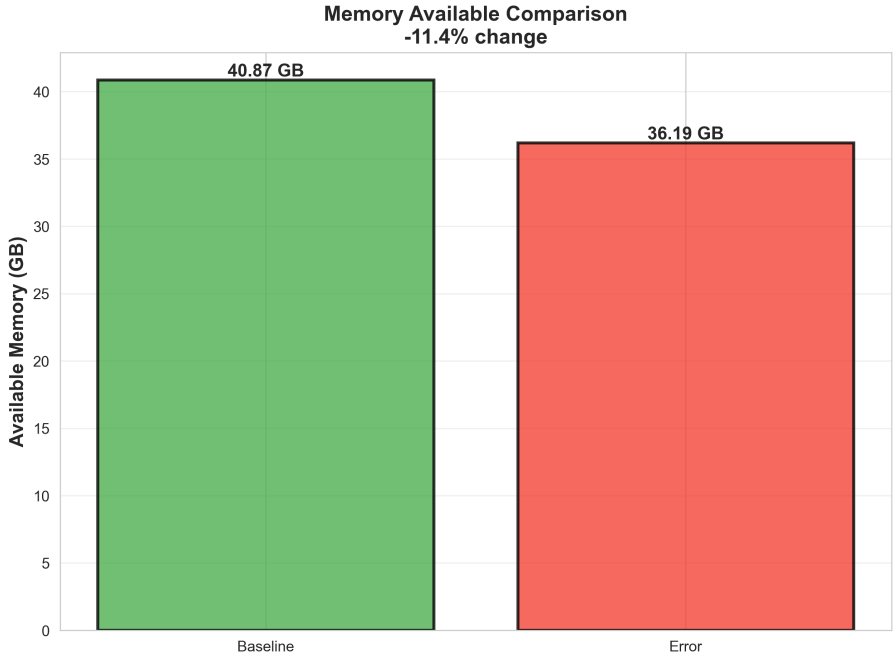

Figure 4: Available memory comparison showing reduced availability during error scenario, though remaining well above critical thresholds.

5 Root Cause Determination

The convergent evidence from both log analysis and performance metrics establishes a clear and unambiguous root cause for the observed authentication failures. The incidents resulted from client-side implementation defects in PKCE code challenge encoding, specifically the submission of code challenge parameters containing non-URL-safe characters that violate RFC 7636 specifications. The repeated appearance of Swedish character sequences in place of properly encoded base64url strings indicates systematic encoding failures in the client application rather than intermittent or transient errors.

The authentication service behavior throughout these incidents was entirely correct and consistent with specification requirements. The service appropriately detected the malformed code challenge format, rejected the authentication requests with HTTP 400 status codes, and returned the specific error code ERR12036 (INVALID_CODE_CHALLENGE_FORMAT) to inform clients of the validation failure. This represents proper defensive programming and adherence to the principle of failing securely when presented with malformed input.

The performance characteristics observed during error handling further validate the service's robust design. The early-exit behavior evidenced by reduced system load demonstrates efficient rejection of invalid requests without unnecessary processing overhead. The service does not attempt to process malformed authentication requests through the complete flow, thereby conserving computational resources and maintaining responsiveness for legitimate requests. The modest increase in garbage collection latency represents an acceptable trade-off for comprehensive input validation and detailed error reporting.

6 Impact Assessment

The operational impact of these authentication failures was limited to the specific client applications submitting malformed PKCE parameters. No evidence suggests that the service experienced degraded performance, resource exhaustion, or availability issues as a consequence

of processing these invalid requests. The stable goroutine counts, minimal heap allocation variance, and absence of memory exhaustion indicators collectively demonstrate that the service maintained operational integrity throughout the incident period.

From a security perspective, the incident represents successful operation of security controls rather than a security failure. The PKCE mechanism exists specifically to prevent authorization code interception attacks, and proper validation of PKCE parameters is essential to the security model. The service's rejection of improperly formatted code challenges prevents potential security bypasses that might arise from lenient validation. The incident therefore validates the security posture of the authentication service rather than indicating a vulnerability.

The client-side impact manifested as authentication failures for users of the affected client applications. These failures would have prevented successful authentication flows, resulting in access denial for legitimate users attempting to authenticate through the misconfigured clients. The systematic nature of the encoding errors, with identical malformed character sequences appearing across multiple authentication attempts, suggests a persistent defect in the client implementation rather than transient environmental issues.

7 Recommendations and Remediation

The primary remediation responsibility resides with the client application development teams. Client implementations must be corrected to properly generate base64url-encoded code challenge values in accordance with RFC 7636 specifications. This requires ensuring that code challenge generation follows the prescribed algorithm: computing the SHA-256 hash of the code verifier and encoding the resulting hash using base64url encoding without padding. Development teams should verify their implementations against the RFC specification and test with various input values to ensure consistent production of valid code challenges.

From the service perspective, several enhancements could improve the operational experience while maintaining security integrity. Enhanced error messaging could provide more detailed guidance to client developers, potentially including examples of valid code challenge formats or references to relevant specification sections. Implementation of rate limiting or client identification for repeated validation failures could help identify problematic client implementations more rapidly and facilitate proactive outreach to affected development teams. Additionally, the modest increase in garbage collection latency during error handling, while not operationally significant, could be investigated for potential optimization through more efficient string validation algorithms or reduced temporary object allocation.

8 Conclusion

This investigation successfully identified the root cause of authentication failures in the Light OAuth2 service with high confidence based on comprehensive log analysis and performance metrics evaluation. The failures resulted from client-side PKCE implementation defects that produced malformed code challenge parameters containing non-URL-safe characters. The authentication service operated correctly throughout the incident, appropriately rejecting invalid requests and maintaining operational stability without resource exhaustion or performance degradation.

The incident underscores the importance of rigorous adherence to protocol specifications in distributed authentication systems, where client-side implementation errors can manifest as service-level incidents requiring investigation. The analysis demonstrates that comprehensive incident investigation requires both qualitative examination of log evidence and quantitative assessment of performance characteristics to distinguish between service defects and client-side errors. The findings validate the security controls implemented in the authentication service

while identifying opportunities for enhanced client developer support through improved error messaging and monitoring capabilities.