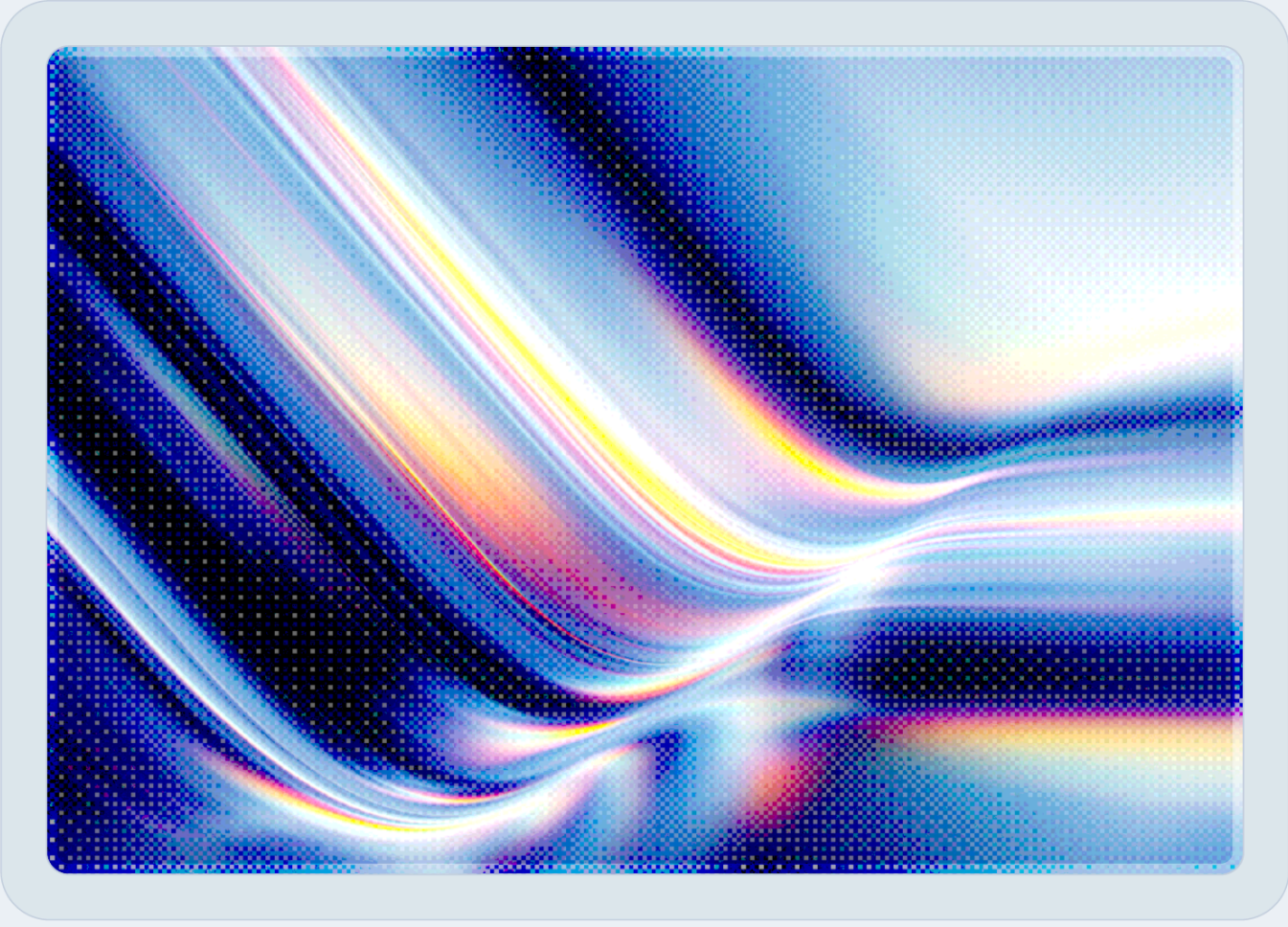
CSAI Foundation | Cloud Security Alliance

Kaltura mwEmbed: Unpatched Deserialization Flaws Enable Unauthenticated RCE

Two Uncoordinated CVEs Threaten Kaltura's Shared Multi-Tenant Video CDN

2026-08-27

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

Two unpatched vulnerabilities in Kaltura's mwEmbed HTML5 video player library—distributed to customers as html5lib—allow a remote, unauthenticated attacker to read arbitrary files from a target server and, in a more severe follow-on step, execute code as the web-server user [1][2]. Tracked as CVE-2026-19913 (arbitrary file read, CVSS 9.1) and CVE-2026-19912 (remote code execution, CVSS 10.0), both flaws originate in the same unsanitized endpoint, `mwEmbedLoader.php`, which deserializes attacker-controlled input without validation [1][2]. Because neither vulnerability requires authentication, a Kaltura session token, or any interaction beyond network access to the endpoint, exploitation requires no specialized skill beyond identifying that the endpoint is reachable, placing it within range of both opportunistic scanning and targeted intrusion attempts.

The disclosure followed an unusually prolonged vendor coordination process, spanning more than five months and multiple escalation channels before resulting in public disclosure without vendor engagement. Independent researcher Gerjan Wemekamp of AndDone first reported the issues to Kaltura in March 2026, escalated through corporate and executive channels over the following months, and ultimately routed the report through a national CERT and CERT/CC after receiving no substantive vendor response [1][2]. CERT/CC's own advisory, VU#308749, states plainly that it was "unable to reach Kaltura to coordinate these vulnerabilities" and lists the vendor's patch status as unknown [2]. As of the August 25, 2026 public disclosure, no patch, hotfix, or official vendor statement had been issued, leaving affected organizations dependent on their own compensating controls [1][2].

The exposure is compounded by Kaltura's shared, multi-tenant content delivery infrastructure. The vulnerable endpoint is reachable not only on self-hosted and dedicated Kaltura deployments but also on the shared CDN hosts that serve multiple customer tenants simultaneously, meaning a single successful exploitation could expose credentials, configuration data, or code-execution access affecting more than one organization at once [1][2]. Given the severity, the low exploitation bar, and the absence of vendor remediation, organizations running Kaltura video infrastructure—embedded video for corporate learning platforms, media companies, higher education, and OTT services among them—should treat this as an active, unmitigated risk requiring immediate compensating controls rather than a routine patch-and-wait advisory.

Background

Kaltura and the mwEmbed Player

Kaltura is a video platform vendor whose technology underpins video management, publishing, and playback for enterprise learning systems, media and entertainment companies, higher education institutions, and OTT streaming services. Kaltura licenses and distributes its player technology both as a component of its own hosted SaaS offering and as an embeddable library, html5lib, that customers integrate directly into self-managed deployments. At the core of that library sits mwEmbed, an HTML5 video player originally built on MediaWiki's embedding framework, which handles player configuration loading, UI customization, and communication with backend Kaltura API services [1]. Because mwEmbed is designed to be embedded across a wide range of customer websites and applications, its loader logic is intentionally network-facing and configurable—a design property that, absent rigorous input validation, becomes the root cause of the vulnerabilities disclosed in August 2026.

A Difficult Coordinated Disclosure

The vulnerability research that produced VU#308749 followed a disclosure timeline stretching more than five months. Wemekamp first reported the flaws to Kaltura's designated security contact on March 23, 2026, then re-sent the report from a corporate email address on April 13 after receiving no acknowledgment [1]. Escalation attempts continued through a LinkedIn outreach to Kaltura's Chief Information Security Officer on May 23 and, by July 2, an approach to a national CERT for assistance [1]. CERT/CC itself formally notified Kaltura on July 8, roughly three and a half months after the initial report, and by the time of public disclosure on August 25 had still received no statement from the vendor [1][2]. Kaltura's published security contact channels—a dedicated security email address and a HackerOne bug bounty program—did not produce a substantive response over this period, according to the researcher and CERT/CC's own account [1][2].

This pattern is not unique to Kaltura, but it illustrates a structural weakness common in coordinated vulnerability disclosure more broadly: security contacts that are unmonitored or under-resourced, bug bounty intake channels that do not reliably reach engineering teams with remediation authority, and no consistent obligation for a vendor to acknowledge, triage, or respond within a defined window. When that breakdown occurs on a shared multi-tenant platform, the consequences of vendor inaction are not confined to the vendor itself—they extend to every customer whose data and infrastructure the platform touches, regardless of how promptly the customer would have applied a fix had one been available.

Security Analysis

CVE-2026-19913: Arbitrary File Read via Unsafe Deserialization

The `mwEmbedLoader.php` endpoint accepts a `ServiceUrl` parameter that is passed, without sanitization, into Kaltura's `KalturaClientBase` PHP client, which deserializes the value using PHP's native `unserialize()` function [1][2]. Because `unserialize()` will process any well-formed serialized string it receives, an attacker can supply a `file://` URI as the `ServiceUrl` value, causing the server to attempt to read and process the referenced local file. Where the deserialization attempt fails on malformed or unexpected input, the resulting PHP error messages frequently echo back file contents in the response, giving the attacker an unauthenticated read primitive into the server's filesystem [1]. Files reachable through this path routinely include Kaltura's `local.ini` configuration file, which stores database credentials, API secrets, and other sensitive configuration values needed to operate the platform [1][2]. CVSS scoring places this vulnerability at 9.1, reflecting its network attack vector, lack of required privileges, and high confidentiality impact even though it does not by itself grant code execution [1].

CVE-2026-19912: Escalation to Remote Code Execution

The second and more severe flaw, CVE-2026-19912 (CVSS 10.0), builds on the same unsafe-deserialization foundation but escalates from file disclosure to full code execution [1][2]. The `uiconf_id` parameter, used by the loader to construct a cache directory path, is likewise processed without sanitization for directory traversal sequences such as `../`. An attacker can combine a crafted `ServiceUrl` value—pointing to a malicious serialized PHP object containing executable payload code—with a `uiconf_id` value that traverses out of the intended cache directory and writes the deserialized object's fields into a web-accessible location [1][2]. Once written, the attacker requests the planted file directly over HTTP, triggering execution of arbitrary PHP code with the privileges of the web-server process [1]. This attack chain requires only that the target deployment use Kaltura's default file-based cache backend, a common configuration, and does not require any authenticated session, API key, or prior reconnaissance beyond identifying that the endpoint is reachable [2].

The technical mechanism—untrusted input flowing into a native deserialization function that reconstructs objects and, downstream, permits arbitrary file writes to code-executable paths—places CVE-2026-19912 in the same vulnerability class CSA's AI Safety Initiative has documented in two related 2026 cases: unsafe pickle deserialization enabling unauthenticated RCE in the LeRobot machine learning framework [3], and a chained SQL-injection-to-deserialization attack against LangGraph's self-hosted

checkpoint persistence layer, an open-source framework for orchestrating stateful AI agents [6]. As CSA's research note on the LeRobot framework's CVE-2026-25874 concluded in a structurally similar case, network proximity to a service does not constitute authorization, and every network-facing component—whether an AI framework or, as here, a video CDN loader—must enforce its own input validation and authentication independent of any assumption that the network perimeter is trusted [3]. The Kaltura case demonstrates that this failure mode is not confined to AI-adjacent software; it is a durable pattern in how legacy PHP deserialization practices interact with modern, internet-facing, multi-tenant service architectures.

The Multi-Tenant Blast Radius

What elevates this disclosure beyond a conventional single-application RCE is Kaltura's infrastructure model. The `mwEmbedLoader.php` endpoint is exposed not only on customer-managed, self-hosted Kaltura instances but also on Kaltura's own shared, multi-tenant CDN hosts, which serve video content and player assets for numerous customer organizations from common infrastructure [1][2]. On a self-hosted deployment, successful exploitation compromises a single organization's data and systems. On the shared CDN, the same vulnerable endpoint may be reachable across tenant boundaries, meaning an attacker who identifies and exploits the flaw against Kaltura's shared infrastructure could potentially access configuration data or execute code affecting multiple unrelated customers whose only connection to one another is that they license video services from the same provider [1][2]. This tenant-crossing exposure illustrates the general shared-responsibility principle CSA has articulated for SaaS platforms: the security boundary customers assume exists between tenants depends entirely on controls the platform operator implements and maintains, not on anything the customer configures on their own [4].

As of the public disclosure date, neither CVE-2026-19912 nor CVE-2026-19913 appeared in the CISA Known Exploited Vulnerabilities catalog, and no confirmed active exploitation had been reported [1]. That should not be read as an indication of low risk. Based on the pattern observed for comparably severe unauthenticated RCEs disclosed without a patch in 2025–2026, weaponization has often followed within days of public disclosure — meaning the absence of a vendor patch here removes the primary defense most organizations would normally rely on.

Recommendations

Immediate Actions

Security teams operating self-hosted Kaltura deployments should inventory every instance exposing the html5lib or mwEmbed component and confirm whether `mwEmbedLoader.php` is reachable from the internet. Where the endpoint is not required for legitimate operation, it should be blocked or removed entirely at the web server or reverse proxy layer; where it must remain reachable, a web application firewall rule restricting or inspecting requests to that path is the most immediately deployable compensating control [1][2]. Organizations that consume Kaltura as a managed SaaS service through the vendor's shared multi-tenant CDN should contact their Kaltura account representative directly to request written confirmation of the tenant's exposure status and the vendor's remediation timeline, given that CERT/CC has publicly stated it could not obtain a vendor response through normal channels [2].

Wherever the endpoint cannot be fully blocked, organizations should enforce a strict allow-list on the `ServiceUrl` parameter that permits only known, legitimate backend Kaltura API hosts, and reject any `uiconf_id` value containing directory traversal sequences before it reaches the application layer [1][2]. Given that CVE-2026-19913 alone can expose database credentials and API secrets stored in `local.ini`, any organization unable to confirm with confidence that the endpoint has been unreachable throughout the disclosure window should rotate those credentials as a precaution [1].

Short-Term Mitigations

Because CVE-2026-19912 depends on the default file-based cache backend accepting attacker-written content into a web-accessible directory, organizations should audit their Kaltura cache configuration and, where feasible, deny PHP execution permissions within cache directories at the web server configuration level—a defense-in-depth control that would blunt the code-execution step even if the underlying deserialization and traversal flaws remain unaddressed [1][2]. Outbound network access from application servers hosting the vulnerable component should be restricted to only the destinations required for legitimate Kaltura API communication, limiting the utility of the `ServiceUrl` parameter as a pathway to attacker-controlled infrastructure.

Security operations teams should add detection logic for anomalous requests to `mwEmbedLoader.php`, particularly requests containing `file://` URI schemes, PHP-serialized object markers, or directory traversal sequences in the `ServiceUrl` and `uiconf_id` parameters, and should monitor for unexpected PHP file creation within cache directories as a potential indicator of

successful exploitation [1][2]. Given the extended and ultimately unsuccessful disclosure process documented in VU#308749, organizations should not assume a vendor patch is imminent and should plan compensating controls to remain in place for an indefinite period.

Strategic Considerations

This disclosure is a case study in a governance gap that security teams should plan for: even established, well-resourced platform vendors do not uniformly maintain the responsive security contact channels that coordinated vulnerability disclosure depends on. Organizations with significant dependence on any single vendor's shared infrastructure should incorporate vendor responsiveness to security disclosures—not just historical patch cadence—into third-party risk assessments, and should establish contractual or contact escalation paths in advance of any incident rather than discovering, mid-crisis, that the vendor's published security channels are unmonitored.

More broadly, the Kaltura case is a reminder that unsafe deserialization remains a persistent, cross-technology vulnerability class. It surfaces with comparable severity and comparable exploitation mechanics whether the vulnerable component is an AI orchestration framework, a robotics learning library, or, as here, a video player loader built on aging PHP deserialization patterns. Application security programs should treat any component that deserializes external input—regardless of whether the surrounding platform is marketed as an AI product—as requiring the same rigor: strict input validation, avoidance of native deserialization functions on untrusted data where safer alternatives exist, and assume-breach architecture that limits the blast radius when a single component fails.

CSA Resource Alignment

This research note connects to several Cloud Security Alliance publications relevant to unsafe deserialization, multi-tenant platform risk, and vendor security coordination.

CSA's Lab Space research note on **LeRobot CVE-2026-25874: Unauthenticated RCE via Pickle** documents a structurally identical vulnerability class—unsafe deserialization of untrusted input over a network-exposed, unauthenticated service, resulting in remote code execution [3]. That note's central conclusion, that network proximity to a service does not constitute authorization and that every network-facing component must independently enforce authentication and input validation, applies directly to the Kaltura mwEmbedLoader.php findings and reinforces that this failure mode recurs across unrelated technology stacks, not only in AI-branded software.

CSA's Lab Space research note on **LangGraph RCE Chain: Checkpointer Flaw Enables Server Takeover** documents a related chained vulnerability—SQL injection in LangGraph's SQLite checkpoint backend combined with unsafe deserialization of checkpoint data—that likewise culminates in unauthenticated remote code execution against a self-hosted, network-facing service [6]. Together with the LeRobot case, it substantiates the broader observation that unsafe deserialization recurs as a distinct root cause across AI orchestration and machine learning tooling, reinforcing that the Kaltura mwEmbedLoader.php flaw sits within a pattern documented across multiple unrelated technology stacks rather than an isolated incident.

CSA's guidance on **Understanding the Shared Responsibility Model in SaaS** is directly relevant to the multi-tenant CDN exposure at the center of this disclosure. The guidance's core argument—that SaaS providers bear primary responsibility for platform and infrastructure security while customers manage access and configuration within their own tenancy—maps onto the Kaltura case in that customers have no configuration lever available to remediate a flaw in the provider's shared loader endpoint; the exposure is entirely within the vendor's control, and the guidance's emphasis on verifying provider security posture rather than assuming it applies with particular force where, as here, the vendor has not responded to coordinated disclosure.

The CSA **AI Controls Matrix (AICM) v1.1** [5], available at **cloudsecurityalliance.org/artifacts/ai-controls-matrix-v1-1**, provides applicable control guidance through its threat and vulnerability management and application security domains even though the vulnerable component itself is not an AI system. AICM's control objectives around secure software development practices, input validation, and vendor security assessment are directly transferable to the deserialization and multi-tenant exposure patterns documented here, and organizations building AI-adjacent products on top of third-party media, CDN, or embedding libraries such as Kaltura's should apply the same control rigor to those dependencies that AICM prescribes for AI components themselves.

References

[1] The Hacker News. "Unpatched Kaltura mwEmbed Flaws Could Let Remote Attackers Read Files and Run Code." The Hacker News, August 26, 2026.

[2] CERT/CC. "VU#308749 - Remote Code Execution and Arbitrary File Read Vulnerabilities in Kaltura Servers." CERT Coordination Center, August 25, 2026.

[3] Cloud Security Alliance. "LeRobot CVE-2026-25874: Unauthenticated RCE via Pickle." CSA Lab Space, April 29, 2026.

[4] Cloud Security Alliance. "Understanding the Shared Responsibility Model in SaaS." Cloud Security Alliance Blog, August 13, 2024.

[5] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, June 22, 2026.

[6] Cloud Security Alliance. "LangGraph RCE Chain: Checkpointer Flaw Enables Server Takeover." CSA Lab Space, June 14, 2026.