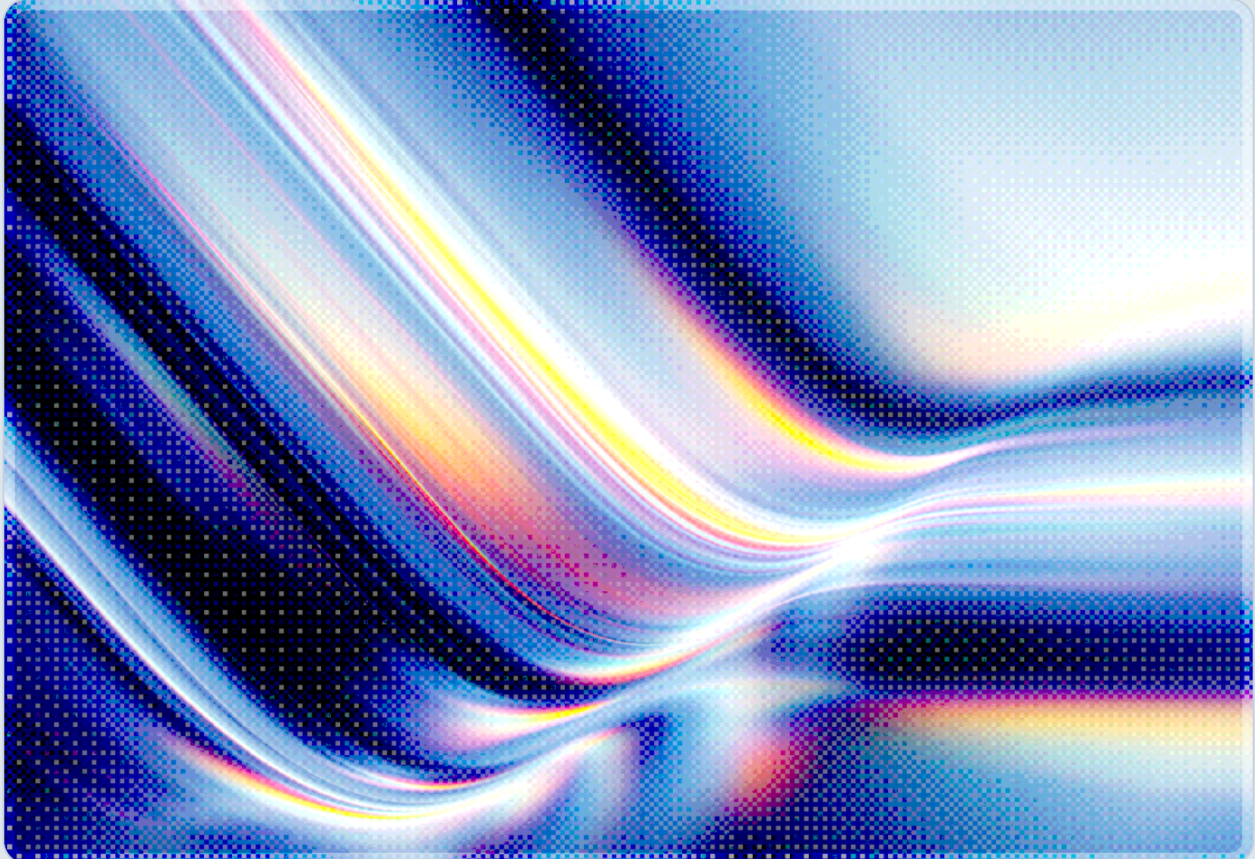


Xinference Actor-Layer RCE: Self-Hosted Inference as Initial Access

CVE-2026-49950: Unauthenticated Cloudpickle Deserialization in Xorbits Xoscar

2026-10-22

 AI-assisted Rapid Research



CSAI

cloud
security
alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

CVE-2026-49950 is a critical unauthenticated remote code execution flaw in Xorbits Xoscar, the actor and RPC framework underneath the Xinference model-serving platform. Every Xoscar release from 0.1.0 through 0.11.2 passes inbound actor-pool traffic to `cloudpickle.loads()` before it checks the message type or authenticates the sender. Anyone who can open a TCP connection to an actor-pool port can therefore run code as the pool process, which the researcher reports commonly runs as root on GPU nodes [1][2]. The vulnerability carries a CVSS v4.0 score of 9.3 [1][4] and a CVSS v3.1 score of 9.8, the latter supplied to the National Vulnerability Database by VulnCheck as a secondary scorer [5].

The CVE was published on October 13, 2026, with no fixed release available. The researcher had reported the flaw on June 8, and the 120-day coordinated-disclosure deadline passed on October 6 without a vendor fix [2]. On October 16, the same day VulnCheck published its technical write-up, the maintainers released Xoscar 0.12.0 and Xinference 4.0.1. These releases add HMAC-SHA256 authentication of actor messages, but the feature is **opt-in and disabled by default** [6][7]. Upgrading the package alone therefore does not close the exposure. As of October 22, 2026, the OSV and VulnCheck records still list no fixed version [4][1], and operators should treat every unconfigured deployment as vulnerable.

Exposure depends heavily on deployment mode. Single-node installations bind the actor pool to a random local port by default. Xinference's distributed (multi-node) mode, by contrast, binds the supervisor and worker pools to `0.0.0.0` by default, which makes any host on the cluster network a potential attacker [2][3]. The REST API's authentication on port 9997 does not protect the actor ports [7][8].

A public exploit module exists, and the researcher reports that exploitation leaves no file on disk [2][3]. We found no public reports of in-the-wild exploitation as of October 22, 2026. Even so, the bug follows the same pattern as Ray's "ShadowRay" vulnerability (CVE-2023-48022, which the vendor disputes), and attackers exploited that flaw for months against exposed AI clusters [10]. Wiz's 2026 honeypot telemetry, together with CSA's analysis of it, also shows attackers building tooling aimed specifically at self-hosted AI components [16][11].

Organizations running Xinference should take four steps now: inventory every Xinference and Xoscar instance, restrict actor-pool ports to trusted cluster peers at the network layer, upgrade every peer to Xoscar 0.12.0 or later, and set a shared `XOSCAR_AUTH_KEY` with a coordinated restart. Any reachable cluster that ran without these controls should be reviewed for signs of compromise.

Background

Xinference (Xorbits Inference) is an open-source platform for serving large language models, embedding models, rerankers, and multimodal models. It exposes an OpenAI-compatible REST API, and organizations often adopt it to run open-weight models on their own GPU hardware instead of calling a hosted provider. Behind the API, Xinference is built on Xoscar, a Python actor framework from the same Xorbits team. Xoscar handles process supervision, remote method invocation, and data transfer between the supervisor and the worker processes that hold model weights. This split matters for security. The HTTP surface that operators see, configure, and sometimes protect with authentication is distinct from the actor transport that carries internal cluster traffic. The researcher notes that the actor port returns no banner and is not typically found by internet-wide scanning [2].

Xoscar serializes actor messages with cloudpickle, an extension of Python's `pickle` that can serialize functions and closures. Python's own documentation warns that unpickling can execute arbitrary code and that pickle data should never be loaded from an untrusted source [15]. Distributed Python frameworks still use pickle-family formats because they make it easy to move arbitrary objects between processes. The safety of the design then depends entirely on ensuring that only trusted peers can reach the deserializer. Before version 0.12.0, Xoscar had no mechanism to enforce that.

Valentin Lobstein (Chocapikk) found the vulnerability and disclosed it through the VulnCheck CVE Numbering Authority [1]. The disclosure timeline is summarized below.

Date (2026)	Event
June 8	Vulnerability reported; CVE requested via VulnCheck CNA [2]
October 6	120-day coordinated-disclosure deadline passes without a fixed release [2]
October 13	CVE-2026-49950 published (CWE-502; CVSS v4.0 9.3) [1][4]
October 16	VulnCheck technical disclosure; Xoscar 0.12.0 and Xinference 4.0.1 released with opt-in actor authentication [2][6][7]
October 17	Researcher's own technical write-up published [3]
October 20	Xoscar 0.12.1 released (functional enhancement only; no change to authentication defaults) [17]

A different Xinference vulnerability is easy to confuse with this one. CVE-2026-61539, published in August 2026, is an unsafe `eval()` in Llama3 tool-call parsing that is reachable through the chat completions API. It affects Xinference versions before 2.7.0 and is fixed in 2.7.0 [9]. Operators who patched that issue in the summer are not protected against CVE-2026-49950. The two flaws sit in different layers and have different remediations.

Security Analysis

Root Cause

The vulnerable path runs through three Xoscar modules. `SocketChannel.recv` in `backends/communication/socket.py` reads an incoming frame and passes the body to the framework's `deserialize` routine. That routine reaches `unpickle_buffers` in `serialization/core.pyx`, which calls `cloudpickle.loads()` on the first body buffer [2] [3]. The only check before this point is in `get_header_length` in `serialization/aio.py`, which validates a one-byte protocol version. That is a framing check and offers no security [3]. The pool has no sender authentication, no restricted unpickler, and no allowlist of message types. Because the pool deserializes the whole message before deciding what kind of message it is, a pickle gadget in the first buffer runs during parsing. The pool then raises an error, but the attacker's command has already executed [2].

In practice this is a minimal attack: one crafted binary frame sent to one TCP port. The attacker needs no credentials or user interaction, does not need a model to be loaded, and does not need a GPU to be present [2]. The researcher reports getting a root reverse shell on a lab pool. They also report that nothing is written to disk and that the attacker can drop the connection right after sending the frame [2] [3]. These exploitation details come from the researcher's disclosure and have not been independently reproduced; the maintainers' release notes confirm the missing authentication but do not describe exploitation themselves [6]. Because the payload runs in memory, file-integrity monitoring and on-disk artifact scanning are unlikely to detect the initial exploitation itself, although follow-on activity such as persistence, cryptominers, or dropped tooling may still leave on-disk traces. Network-flow records and process-ancestry telemetry are better sources for detecting the initial compromise.

Exposure by Deployment Mode

How exposed a deployment is depends mostly on how Xinference was launched. The table below summarizes the default exposure for the common modes, based on the researcher's analysis, the project documentation, and Kubernetes networking defaults.

Deployment mode	Actor-pool binding (default)	Who can reach the vulnerable sink
Single-node (<code>xinference-local</code>) on a host	Random local port in the 10000–65535 range [2]	Local processes and users on the same host; the network if the host is reconfigured
Single-node in Docker	Documentation launches <code>xinference-local -H 0.0.0.0</code> inside the container; with that host argument the supervisor actor pool also binds to all interfaces, as the documented startup log ("Xinference supervisor 0.0.0.0:...started") shows [3][8]	Other containers and hosts that can route to the container network, depending on port publishing and network mode
Distributed (supervisor + workers)	<code>0.0.0.0</code> by default, controlled by <code>XINFERENCE_DEFAULT_DISTRIBUTED_HOST</code> [2][3]	Any host on the cluster network segment, and beyond it if segmentation is weak
Kubernetes / orchestrated clusters	Pod IPs on the cluster network	Any pod not restricted by a NetworkPolicy, since pods

Deployment mode	Actor-pool binding (default)	Who can reach the vulnerable sink
		accept traffic from all sources by default, including compromised neighbor workloads [19]

The distributed case carries the most risk. In multi-node deployments the supervisor and workers must talk to each other, so the actor ports have to be reachable on the network, and default bindings expose them on all interfaces. The organizations most likely to run distributed Xinference, those serving models at scale on shared GPU infrastructure, are therefore likely to face the largest exposure, particularly where GPU networks are flat. The researcher does not provide counts of internet-exposed instances, and the lack of a banner makes such counts hard to produce [2]. We cannot quantify the public exposure. We expect most of the real risk to come from internal reachability rather than direct internet exposure: a foothold anywhere on a flat data-science or GPU network may be enough to take over the inference tier.

Why the Fix Does Not Close the Gap by Itself

Xoscar 0.12.0 adds HMAC-SHA256 authentication over both the socket and UCX transports, and each message is verified before Python objects are deserialized [6][7]. The design has operational properties that limit how quickly it will reduce real-world exposure. Authentication is disabled unless a nonempty `XOSCAR_AUTH_KEY` is set. When the key is unset, the wire format matches 0.11.2, so an upgraded but unconfigured cluster remains fully exploitable [6][7]. Keyed and unkeyed peers cannot talk to each other, and there is no mixed-mode rolling upgrade. Turning authentication on, and rotating the key later, requires a coordinated restart of every supervisor, worker, sub-pool, and external client [6]. That restart means downtime for every peer, which is a real cost for teams serving production inference traffic and may push some operators to postpone enabling authentication.

The maintainers also say that authentication provides message integrity but does not encrypt traffic or prevent replay of captured messages, and that every holder of the key is a trusted peer [6][7]. The shared secret therefore becomes a high-value credential. Anyone who obtains it, for example from an environment variable in a compromised container, a Helm values file, or a CI log, is back in the pre-patch

position. The project's guidance still tells operators to restrict actor-port access to trusted cluster peers with network controls [8]. Network isolation remains the primary control, and the HMAC layer is an additional defense on top of it.

Finally, authentication has a performance cost. Authenticated GPU transfers stage device buffers in host memory for hashing, which adds CPU and host-memory overhead [6]. Teams that run tight on host memory may see this as a reason to leave authentication off. Operators should measure the cost and decide explicitly rather than accept the insecure default.

Impact: The Inference Node as a Beachhead

Code execution on an inference node usually gives an attacker far more than one compromised process. The researcher reports that the pool commonly runs as root on GPU nodes, and their own demonstration produced a root shell [1][2]. That position gives the attacker access to loaded model weights (which may include proprietary fine-tunes), to the prompts and completions passing through the service, and to the GPUs themselves [3]. Inference hosts also tend to hold credentials: model-hub tokens, object-storage keys for weight repositories, and cloud instance roles that may be broadly scoped so models can be pulled at startup. From there an attacker may be able to move into data stores, training pipelines, or the cloud control plane.

The precedent is well documented. Ray's CVE-2023-48022 is also a missing-authentication flaw in a distributed AI compute framework that allowed remote code execution [10]. Anyscale disputed the CVE on the grounds that Ray is intended to run only inside trusted networks, and the CVE remains marked as disputed [10]. That position illustrates the design assumption at issue here: reachability is treated as authorization, and exposure becomes the operator's responsibility. Oligo Security reported in March 2024 that attackers had been exploiting the flaw for roughly seven months, a period that included the months after the CVE was publicly disclosed in late 2023. The compromised clusters were used for cryptomining, secret theft, and exfiltration of data from live AI workloads [10]. Ray has remained a target since then; CSA's August 2026 analysis covered a separate Ray RCE, CVE-2025-62593, added to CISA's Known Exploited Vulnerabilities catalog after active exploitation [18].

Recent telemetry points the same way. Wiz reported that 90% of the cloud environments it observed run self-hosted AI software and 63% self-host models [16]. CSA's September 2026 analysis of Wiz's accompanying honeypot telemetry found that attackers going after self-hosted AI infrastructure use purpose-built tooling written for AI-stack internals, not generic scanners [11]. Together, these points suggest that a pre-authentication RCE in a model-serving framework with a public exploit module is a plausible near-term target, even though no in-the-wild exploitation has been reported yet.

Recommendations

Immediate Actions

The first step is to find every instance, including unofficial ones. Xinference is often installed by individual data scientists or research teams rather than by central IT. Inventories should therefore check Python environments (`pip show xoscar xinference`), container images based on `xprobe/xinference`, and Kubernetes workloads, not only managed asset lists. Any Xoscar version at or below 0.11.2 is vulnerable [1]. A version at or above 0.12.0 is vulnerable too until `XOSCAR_AUTH_KEY` is set on every peer [6].

Next, restrict network reachability to the actor ports before attempting the upgrade. Supervisor and worker actor ports should be reachable only from the other members of the same cluster. Use host firewalls, cloud security groups, or Kubernetes NetworkPolicy, as the researcher and project recommend [2][8]. For single-node deployments, confirm that the pool is not bound to an externally reachable interface and that container networking does not expose it to neighboring workloads. Treat any host that can reach an actor pool as able to execute code on it [2].

Then upgrade every peer to Xoscar 0.12.0 or later, or Xinference 4.0.1 or later, and enable authentication with a coordinated restart. After upgrading Xinference, verify the installed Xoscar version directly with `pip show xoscar`, because upgrading Xinference alone may not upgrade Xoscar where dependencies are pinned separately. Generate a high-entropy shared secret, distribute it through a secrets manager rather than plaintext configuration, and set it on the supervisor, all workers, and any external clients [6][8]. Separately, confirm that Xinference is at 2.7.0 or later to address CVE-2026-61539 in the API layer [9].

Short-Term Mitigations

Clusters that were reachable from untrusted segments before these controls were applied should get a focused compromise review. Because the initial exploitation leaves no file on disk [2], the review should focus on network flow logs showing connections to actor ports from hosts outside the cluster, unexpected child processes (shells, `curl`, `wget`, interpreters) spawned by Xinference or Xoscar processes, new outbound connections from GPU nodes, and unusual GPU utilization that might indicate cryptomining. Disk-based checks for persistence mechanisms and dropped tooling remain worthwhile for follow-on activity. Credentials available to the inference hosts, including model-hub tokens, storage keys, and instance-role sessions, should be rotated wherever compromise cannot be ruled out.

Hardening the hosts themselves limits the damage of any future compromise. Run inference processes as unprivileged users, in containers without host-level privileges, so that a compromised pool does not automatically mean root on the GPU node. Scope cloud identities attached to inference hosts down to read-only access for the specific model artifacts they need. Restrict egress from inference nodes to known model registries and internal services. That limits both data exfiltration and the reverse-shell pattern shown in the researcher's demonstration [2].

Strategic Considerations

CVE-2026-49950 fits a recurring pattern. Distributed AI compute frameworks are designed to run code across nodes, and they can treat network reachability as authorization. Ray, and now Xoscar, follow this pattern, and the vendor dispute over Ray's CVE shows that some maintainers regard it as intended behavior [2][10]. Organizations should treat the internal transports of any model-serving, orchestration, or training framework as remote-code-execution interfaces by design. Their security reviews should ask how inter-node traffic is authenticated, not only whether the HTTP API is protected. Procurement and internal platform standards should require these frameworks to authenticate by default, or should document compensating network controls when they cannot.

The disclosure history also bears on risk acceptance for open-source AI infrastructure. The vulnerability went past a 120-day disclosure window, and the fix shipped as an opt-in feature that preserves the old wire format when no key is set [2][6]. Teams building on fast-moving AI frameworks should budget for the possibility that critical fixes will arrive late or require configuration changes to take effect. Network segmentation of AI workloads into dedicated zones, with explicit allow rules between components, gives protection that does not depend on any single project's patch cadence. Finally, self-hosted inference should be held to the same patch and segmentation standards as other production systems. CSA's analysis of 2026 attack telemetry against AI gateways and agent frameworks reached the same conclusion [11].

CSA Resource Alignment

This vulnerability belongs to a series of pre-authentication flaws in self-hosted AI infrastructure that CSA has tracked through 2026. CSA's research note [90 Days of Attacks on AI Infrastructure: A Honeypot View](#) (September 2026) is the most direct companion to this analysis [11]. Its finding that attackers are building tooling specific to AI-stack internals, and its recommendation to treat AI components as production systems with patch cadence, segmentation, and identity controls, apply directly to Xinference clusters. The Xoscar actor port is the kind of non-HTTP internal surface that such purpose-built tooling could

target. CSA's [Ray Framework RCE Under Active Exploitation Joins CISA KEV](#) (August 2026) is the closest precedent at the framework level, documenting active exploitation of another distributed AI compute framework whose internal interfaces assume a trusted network [18].

CSA's [LangGraph RCE Chain: Checkpointer Flaw Enables Server Takeover](#) (June 2026) analyzed another case where unsafe deserialization in a self-hosted AI framework layer led to code execution [12]. Read together, the LangGraph checkpointer and the Xoscar transport show that object-deserialization formats remain a recurring root cause in AI frameworks, and that the fix often depends on deployment configuration as well as package versions. CSA's analysis of the LiteLLM AI gateway chain, [LiteLLM AI Gateway: Active Exploitation via MCP Injection](#), adds a related lesson [13]. In that chain, an authentication assumption at the HTTP layer failed and exposed a code-execution surface. Xinference shows a variant of the same problem: its documentation notes that HTTP API authentication does not extend to the actor ports at all [8].

For control mapping, CSA's [AI Controls Matrix \(AICM\) v1.1](#) provides the governing framework [14]. The recommendations in this note map to its Threat and Vulnerability Management domain (inventory and timely remediation of AI infrastructure components), its Application and Interface Security domain (secure design of inter-component interfaces, including deserialization safety), its Infrastructure Security domain (network segmentation of AI workloads), and its Identity and Access Management domain (least-privilege scoping of credentials available to inference hosts and protection of shared secrets such as `XOSCAR_AUTH_KEY`).

References

- [1] VulnCheck. "[Xorbits Xoscar 0.11.2 Unauthenticated Deserialization RCE via Actor Pool](#)." VulnCheck Advisories, October 13, 2026.
- [2] VulnCheck (Valentin Lobstein). "[Xorbits Xoscar: Unauthenticated RCE via Cloudpickle Deserialization in the Actor Pool](#)." VulnCheck Blog, October 16, 2026.
- [3] Valentin Lobstein (Chocapikk). "[Xorbits Xoscar: Unauthenticated RCE via Cloudpickle Deserialization in the Actor Pool](#)." Chocapikk's Cybersecurity Blog, October 17, 2026.
- [4] OSV. "[CVE-2026-49950](#)." Open Source Vulnerabilities Database, modified October 21, 2026.
- [5] NIST. "[CVE-2026-49950 Detail](#)." National Vulnerability Database, October 2026.
- [6] Xorbits. "[Xoscar Release v0.12.0](#)." GitHub, October 16, 2026.
- [7] Xorbits. "[Xinference Release v4.0.1](#)." GitHub, October 16, 2026.
- [8] Xorbits. "[Using Xinference](#)." Xinference Documentation, accessed October 22, 2026.
- [9] GitLab. "[CVE-2026-61539: Xinference Remote Code Execution via Unsafe eval\(\) in Llama3 Tool-Call Parsing](#)." GitLab Advisory Database, August 21, 2026.
- [10] Oligo Security. "[ShadowRay: First Known Attack Campaign Targeting AI Workloads Exploited In The Wild](#)." Oligo Security Blog, March 26, 2024.
- [11] Cloud Security Alliance. "[90 Days of Attacks on AI Infrastructure: A Honeypot View](#)." CSA AI Safety Initiative, September 1, 2026.
- [12] Cloud Security Alliance. "[LangGraph RCE Chain: Checkpointer Flaw Enables Server Takeover](#)." CSA AI Safety Initiative, June 14, 2026.
- [13] Cloud Security Alliance. "[LiteLLM AI Gateway: Active Exploitation via MCP Injection](#)." CSA AI Safety Initiative, June 13, 2026.
- [14] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1](#)." Cloud Security Alliance, June 22, 2026.
- [15] Python Software Foundation. "[pickle – Python object serialization](#)." Python 3 Documentation, accessed October 22, 2026.

- [16] Wiz Threat Research (Yaara Shriki). "[Inside 90 Days of Attacks on AI Infrastructure](#)." Wiz Blog, August 27, 2026.
- [17] Xorbits. "[Xoscar Release v0.12.1](#)." GitHub, October 20, 2026.
- [18] Cloud Security Alliance. "[Ray Framework RCE Under Active Exploitation Joins CISA KEV](#)." CSA AI Safety Initiative, August 18, 2026.
- [19] The Kubernetes Authors. "[Network Policies](#)." Kubernetes Documentation, accessed October 22, 2026.