The Fourth Pickle: Unauthenticated Remote Code Execution in SGLang's Disaggregated Diffusion Server

CVE	CVE-2026-93088
CVSS	AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
CERT/CC	VU#727584
GitHub Security Advisory	GHSA-8374-wrr5-7q7f

2026/07/03: Reported via GitHub Security Advisory

2026/08/14: Reported to CERT/CC VINCE (VU#727584)

2026/09/17: CERT/CC verified the vulnerability and assigned CVE-2026-93088

2026/09/22: Submitted supplementary findings; drafted this report

This article discusses an unauthenticated remote code execution (RCE) vulnerability in SGLang, tracked by CERT/CC as VU#727584 and by GitHub as GHSA-8374-wrr5-7q7f (CVE-2026-93088).

In short, any host that can establish a TCP connection to the ZeroMQ frontend of a `DiffusionServer` head node can execute arbitrary code inside that process. This occurs because the final frame of incoming multipart messages is passed directly to `pickle.loads()` prior to any validation or authentication.

One notable aspect of this vulnerability is the timing of its introduction. SGLang had previously received three CVEs in March 2026 for this exact vulnerability class — unauthenticated ZeroMQ sockets deserializing attacker-controlled pickle payloads. However, the vulnerable file discussed here was introduced roughly five weeks later as part of a new disaggregated diffusion feature, containing a new [illegible]ls sink. As of writing, the vulnerable code remains present in the latest release and [illegible]n branch.

Skip to content

Following this initial finding, further audit of the surrounding codebase revealed several related security issues across the runtime, resulting in ten distinct findings. All findings have been dynamically verified against the upstream code.

Background

SGLang is a widely used serving framework for large language and multimodal models. Its multimodal generation runtime (`sglang.multimodal_gen`) executes text-to-video and text-to-image pipelines, coordinating work between processes via ZeroMQ sockets that transmit pickled Python objects.

Python's `pickle` module reconstructs objects by executing a stream of opcodes. When unauthenticated network data is passed directly to `pickle.loads` , any callable specified by the sender is invoked in the context of the receiving process, directly resulting in arbitrary code execution.

This issue has a documented history in the SGLang codebase:

- **CVE-2026-3059** (GHSA-rgq9-fqf5-fv58, published 2026-03-12): Unauthenticated RCE via the multimodal ZMQ broker in `scheduler_client.py` (≤ 0.5.9).
- **CVE-2026-3060** (GHSA-jx93-g359-86wm, published 2026-03-12): Unauthenticated RCE via the encoder disaggregation module in `encode_receiver.py` (≤ 0.5.9).
- **CVE-2026-7301** (VU#777338): Unauthenticated RCE via a ROUTER socket sink in the multimodal scheduler path.

To remediate earlier instances, the project introduced `safe_pickle_loads` (a restricted SafeUnpickler implementation) to replace bare `pickle.loads` calls in specific modules, such as `encode_receiver.py` .

However, on 2026-04-16, commit `9da998a8` (PR #21701, "[diffusion] feat: disaggregated diffusion") introduced `python/sglang/multimodal_gen/runtime/disaggregation/orchestrator.py` . This feature separates diffusion pipelines into distinct worker roles (Encoder, Denoiser, Decoder) coordinated by a `Server` head node. The head node accepts incoming client requests over a ZeroMQ ROUTER socket.

Skip to content

Investigation and Request Tracing

Given the history of deserialization issues in the codebase, I audited network-facing paths in the then-current release (0.5.14) to determine whether any raw `pickle.loads` sinks remained. This revealed the unauthenticated deserialization sink in the newly introduced disaggregated orchestrator.

Tracing the request lifecycle illustrates how the sink is reached.

Disaggregated mode is enabled using the `--disagg-role` flag. The head node is launched as follows:

```
sglang serve --model-path Wan-AI/Wan2.1-T2V-14B-Diffusers \
    --disagg-role server \
    --encoder-urls  "tcp://10.0.0.1:19000" \
    --denoiser-urls "tcp://10.0.0.2:19001" \
    --decoder-urls  "tcp://10.0.0.3:19002" \
    --host 0.0.0.0 --port 30000 \
    --scheduler-port 19655
```

This invokes `launch_disagg_server()` in `python/sglang/multimodal_gen/runtime/launch_server.py`, where the frontend endpoint is constructed:

```
host = server_args.host or "127.0.0.1"
base_port = server_args.scheduler_port          # default: 5555
...
frontend_endpoint = f"tcp://{host}:{base_port}"
```

Here, `host` is taken directly from the CLI `--host` parameter.

Next, `DiffusionServer._event_loop()` in `orchestrator.py` binds the frontend socket:

```
frontend, _ = get_zmq_socket(
    context, zmq.ROUTER, self._frontend_endpoint, bind=True
```

Skip to content

The helper function `get_zmq_socket()` (in `runtime/utils/common.py`) configures basic socket options and binds to the specified address. However, it does not enable transport security such as ZeroMQ CURVE encryption or ZAP authentication, nor does the application implement an authentication handshake or token verification. As a result, the ROUTER socket accepts connections from any client.

When incoming data is received, the server invokes `_handle_client_request(frontend)`:

```
def _handle_client_request(self, frontend: zmq.Socket) -> None:
    try:
        parts = frontend.recv_multipart(zmq.NOBLOCK)
    except zmq.Again:
        return

    if len(parts) < 3:
        return

    client_identity = parts[0]
    payload = parts[-1]

    try:
        reqs = pickle.loads(payload)   # <-- the sink
    except (pickle.UnpicklingError, EOFError):
        ...
```

The handler checks only the frame count before passing the final frame directly to `pickle.loads()`. Because validation of the payload structure occurs only after deserialization, a connecting peer can execute arbitrary code inside the server process by sending a crafted pickle payload.

The `--host` Parameter and Bind Behavior

When `--host` is omitted, the frontend binds to `127.0.0.1`, restricting exposure to local processes. However, the CLI help for `--host` states:

Skip to content

```
  --host   Host for the HTTP API server.
```

While documented as the bind address for the HTTP API server, this flag simultaneously controls the bind address for internal ZeroMQ sockets (including the frontend ROUTER socket and result PULL sockets at `scheduler_port + 1..3`). There is no separate CLI flag to configure the ZeroMQ bind address independently.

Furthermore, the official deployment documentation (`docs/docs/sglang-diffusion/disaggregation.mdx`) recommends passing `--host 0.0.0.0` in both single-machine and multi-machine deployment guides. Operators following the official documentation will inadvertently expose the unauthenticated ZeroMQ deserialization endpoint on all network interfaces.

Verification on Real Packages

To confirm the vulnerability, I tested the code against an installed environment (`sglang==0.5.17`, `pyzmq`, and minimal dependencies), stubbing heavy GPU/HTTP imports to allow headless execution on standard environments. Inspecting the live socket created by `DiffusionServer` confirmed the absence of authentication:

```
zmq_type        ROUTER
CURVE_SERVER    0
PLAIN_SERVER    0
ZAP_DOMAIN      ''
```

A non-destructive proof-of-concept client was constructed to connect as an unauthenticated DEALER peer and transmit a harmless marker gadget:

```
class Gadget:
    def __reduce__(self):
        return (print, ("!!! ARBITRARY CODE EXECUTED VIA pickle.loads() ON THE ROUTER
```

Upon message delivery, the marker executed in the server process:

Skip to content

```
!!! ARBITRARY CODE EXECUTED VIA pickle.loads() ON THE ROUTER SOCKET !!!
```

```
PoC marker: this process ran attacker-supplied code.
```

The vulnerable code path is present in all versions from **v0.5.11** through **v0.5.20** and the current `main` branch.

Additional Findings Across the Runtime

An audit of adjacent components revealed several additional vulnerabilities across the disaggregated fabric and related services. All of these behaviors were dynamically verified against `main` (@ `790551c`).

Wildcard Bind and Unauthenticated pickle.loads in Encoder Workers

Worker instances (`--disagg-role encoder|denoiser|decoder`) bind a work PULL socket to receive dispatched jobs. The endpoint is computed in `DisaggServerArgsMixin` :

```
def derive_pool_work_endpoint(self) -> str:
    return format_tcp_endpoint("0.0.0.0", self.scheduler_port, "pool_work_endpoint")
```

The bind address `0.0.0.0` is hardcoded. It does not consult `--host` , and no CLI flag exists to restrict it to specific interfaces.

In the encoder worker (`disaggregation/scheduler_mixin.py` in `_disagg_encoder_step`), incoming frames from this socket are passed directly to `pickle.loads()` :

```
frames = self._pool_work_pull.recv_multipart()
pickled_req = frames[-1]
reqs = pickle.loads(pickled_req)
```

Skip to content r binds the head node to `127.0.0.1` , every encoder worker node remains bound to `0.0.0.0` and vulnerable to direct unauthenticated RCE over the network. Notably, denoiser and

decoder workers use JSON-based transfer messages for this channel; only the encoder path relies on bare `pickle.loads`.

Residual Sink in the Monolithic Scheduler

In the monolithic scheduler (`managers/scheduler.py` in `Scheduler.recv_reqs()`), data received over the ROUTER socket bound to `tcp://{--host}:{scheduler_port}` is similarly passed to `pickle.loads(payload)`. This path corresponds to CVE-2026-7301 (VU#777338), which remains unpatched in current releases and `main`.

Lack of Authentication Across the Distributed Fabric

Other channels in the distributed fabric — including the result PULL endpoints and transfer control channels — likewise lack peer authentication. While these channels deserialize messages using JSON rather than pickle, parameters such as `dest_session_id` and `dest_addr` in `transfer_push` are passed directly to the underlying RDMA transfer engine without validation, and `transfer_ready` applies `scalar_fields` directly via `setattr(req, key, value)`, permitting attribute injection into request objects.

Additionally, the broker in `scheduler_client.py` uses raw `pickle.loads` on both incoming requests and incoming responses, exposing both the broker and client processes to deserialization attacks if the local socket is accessed.

HTTP API Authentication and Additional Surface

The FastAPI HTTP server in `multimodal_gen` does not implement an authentication layer (unlike the `srt` runtime, which provides `--api-key` and `--admin-api-key`). Administrative endpoints such as `POST /update_weights_from_disk` (which accepts arbitrary Hugging Face repository IDs) and `POST /release_memory_occupation` are mounted unconditionally by default.

Skip to content

e and video endpoints fetch user-supplied URLs (`image_url`, `reference_url`) server-side with `follow_redirects=True` and no domain or IP allowlist, resulting in blind SSRF.

Two additional vulnerabilities were identified:

1. Unauthenticated Arbitrary File Write (CWE-22 → RCE)

In `multimodal_gen/runtime/entrypoints/openai/utils.py`, multipart file uploads on endpoints such as `POST /v1/videos` and `/v1/actions` construct target paths using unsanitized filenames: `os.path.join(uploads_dir, f"{request_id}_{filename}")`. The server writes incoming raw bytes directly to this path. By supplying path traversal sequences (`../`), an unauthenticated attacker can write arbitrary files outside `uploads_dir` with the permissions of the server user (often root in containerized deployments). This was dynamically verified by writing a test marker to `/tmp` via `save_image_to_path`.

2. NumPy Object Array Deserialization via WebSocket

Realtime WebSocket endpoints (`/v1/actions/realtime` , `/openpi/policy`) parse msgpack payloads and pass attacker-controlled fields to `np.ndarray(buffer=data, dtype=np.dtype(dtype), shape=shape)`. Specifying `dtype="O"` constructs an object-pointer array over raw input bytes, providing a memory corruption primitive.

Unauthenticated Deserialization in the Core LLM Runtime (srt)

The core LLM serving runtime (`srt`) exhibits a similar pattern in multi-node configurations. When `--enable-dp-attention` is used in a multi-node deployment, `PortArgs.init_new` binds internal ZeroMQ sockets to TCP (`tcp://{dist_init_host}:{port}`). Because `SGLang_USE_PICKLE_IPC` defaults to `True`, sockets such as the tokenizer manager's result PULL socket execute raw `recv_pyobj()` (pickle deserialization) on unauthenticated TCP traffic. Additionally, the multi-node rendezvous layer (`TCPStore`) deserializes stored values via `recv_obj` / `broadcast_obj` without authentication.

Skip to content

Arbitrary code execution in `DiffusionServer` compromises the central control plane of the deployment. The process holds model configurations, API keys, request queues, and active connections to all worker nodes. Compromising the head node enables pivoting to attached worker instances. Furthermore, as noted above, encoder worker nodes can be compromised directly via their hardcoded `0.0.0.0` bind, and the HTTP service provides an independent arbitrary file write primitive.

Reporting History and Remediation

This issue was reported via GitHub Private Vulnerability Reporting (GHSA-8374-wrr5-7q7f) on 2026-07-03, and CERT/CC opened case VU#727584 on 2026-08-14. To facilitate prompt and coordinated resolution, subsequent communication with the maintainers was entrusted to CERT/CC (VU#727584), who independently verified the vulnerability and assigned CVE-2026-93088, leading to coordinated public disclosure.

Recommended remediation steps:

- Replace pickle deserialization across network and IPC boundaries with safe formats such as JSON or validated msgpack.
- As an immediate mitigation, apply `safe_pickle_loads` (SafeUnpickler) to all remaining deserialization sites.
- Implement ZeroMQ CURVE encryption and ZAP authentication for all cluster-facing sockets.
- Decouple the ZeroMQ bind address from the HTTP `--host` flag, defaulting internal sockets to `127.0.0.1`.
- Remove hardcoded `0.0.0.0` binds in worker socket definitions (`derive_pool_work_endpoint`) and provide configuration options to bind specific interfaces.
- Add authentication to the `multimodal_gen` HTTP server, sanitize upload filenames, and enforce validation on media fetch URLs.

Skip to content

ıvailable, operators should enforce network-level filtering to restrict access to `scheduler_port` and associated ports to trusted IP addresses across all cluster nodes.

Timeline

Date	Event
2026-03-12	CVE-2026-3059 / CVE-2026-3060 published (ZMQ pickle RCE family, ≤ 0.5.9)
2026-04-16	Commit `9da998a8` (PR #21701) introduces `orchestrator.py` with raw `pickle.loads`
2026-05-05	v0.5.11 released (first affected release)
2026-07-03	Reported via GitHub Private Vulnerability Reporting (GHSA-8374-wrr5-7q7f)
2026-08-14	CERT/CC receives report and opens VU#727584
2026-08-20	Re-verified on v0.5.17 and `main`; evidence package submitted
2026-09-17	CERT/CC verifies the issue and assigns CVE-2026-93088
2026-09-18	v0.5.20 released (still affected)
2026-09-22	Supplementary findings verified and submitted to CERT/CC

Reporter: Keita Sode (SYZD Research). Thank you to CERT/CC for coordinating the case. The verification proof-of-concept performs only harmless `print()` calls inside the process, without spawning shells, creating files, or initiating external network connections.

blog miso