

PR #48583 完整报告

vllm-project/vllm

[Bugfix][Security] Fix concurrent sparse invariant race bypassing CVE remediation

合并时间: 2026-07-14 19:08

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/48583>

执行摘要

- 一句话: 修复并发竞态绕过稀疏张量安全检查
- 推荐动作: 值得精读, 展示了将全局状态并发访问串行化的典型模式。对于处理 PyTorch 等库的进程级标志的团队有参考价值。

功能与动机

PyTorch 的 `torch.sparse.check_sparse_tensor_invariants()` 操作一个进程全局标志 (`save/enable/restore`)。当多个嵌入加载操作在线程池上并发运行时, 一个上下文可能在另一个线程仍在其保护区内时恢复标志为 `False`, 从而绕过不变量检查。这是一个真实的 CVE (CVE-2025-62164) 修复后仍存在的绕过路径, 需要通过线程锁串行化所有对该上下文管理器的访问来彻底消除竞态。

实现拆解

1. 新增线程安全包装器: 在 `vllm/utils/sparse_utils.py` 中创建 `check_sparse_tensor_invariants_threadsafe` 上下文管理器, 内部组合 `threading.Lock` 与 `torch.sparse.check_sparse_tensor_invariants()`, 确保同一时间只有一个线程能进入全局标志的保存 / 恢复周期。
2. 替换所有调用点: 将 `vllm/multimodal/media/audio.py`、`vllm/multimodal/media/image.py`、`vllm/renderers/embed_utils.py` 中所有原始 `with torch.sparse.check_sparse_tensor_invariants()` 语句替换为 `with check_sparse_tensor_invariants_threadsafe()`, 并添加相应导入。
3. 新增并发安全测试: 创建 `tests/renderers/test_sparse_tensor_concurrent_race.py`, 包含:
 - `TestNegativeControlWithoutRace`: 验证即使无并发, 恶意稀疏张量也会被正确拒绝。
 - `TestConcurrentRaceProtection`: 使用 `threading.Barrier` 强制构造精确的交错场景, 证明锁机制能彻底阻止竞态绕过。

关键文件:

- `vllm/utils/sparse_utils.py` (模块 工具函数; 类别 `source`; 类型 `core-logic`; 符号 `check_sparse_tensor_invariants_threadsafe`): 核心修复, 实现线程安全包装器
- `vllm/multimodal/media/audio.py` (模块 音频输入; 类别 `source`; 类型 `dependency-wiring`): 音频嵌入加载路径, 替换为线程安全包装器

- vllm/multimodal/media/image.py (模块 图像输入; 类别 source; 类型 dependency-wiring) : 图像嵌入加载路径, 替换为线程安全包装器
- vllm/renderers/embed_utils.py (模块 嵌入加载; 类别 source; 类型 dependency-wiring) : 提示嵌入加载路径, 替换为线程安全包装器
- tests/renderers/test_sparse_tensor_concurrent_race.py (模块 并发测试; 类别 test; 类型 test-coverage; 符号 _encode_tensor, _create_malicious_sparse_tensor, _create_valid_dense_tensor, model_config) : 新增完整测试套件, 覆盖无并发拒绝和并发竞态防护

关键符号: check_sparse_tensor_invariants_threadsafe,
test_malicious_sparse_rejected_by_prompt_loader,
test_malicious_sparse_rejected_by_image_loader,
test_malicious_sparse_rejected_by_audio_loader,
test_forced_interleaving_still_rejects_invalid

关键源码片段

vllm/utils/sparse_utils.py

核心修复, 实现线程安全包装器

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
"""
Thread-safe wrapper for sparse tensor invariant validation.

PyTorch's `torch.sparse.check_sparse_tensor_invariants()` context manager
manipulates a process-global flag (save/enable/restore). When multiple
embedding-load operations run concurrently on a thread-pool executor, one
context can restore the flag to `False` while another thread is still inside
its guard, bypassing the invariant check.

All call sites MUST use `check_sparse_tensor_invariants_threadsafe()`
which serializes access behind a lock.
"""

import contextlib
import threading

import torch

# 进程级锁, 确保同一时间只有一个线程操作全局标志
_SPARSE_LOAD_LOCK = threading.Lock()

@contextlib.contextmanager
def check_sparse_tensor_invariants_threadsafe():
    # 组合锁与原始上下文管理器, 进入时加锁, 退出时自动释放
```

```
with _SPARSE_LOAD_LOCK, torch.sparse.check_sparse_tensor_invariants():
    yield
```

tests/renderers/test_sparse_tensor_concurrent_race.py

新增完整测试套件，覆盖无并发拒绝和并发竞态防护

```
# SPDX-License-Identifier: Apache-2.0
```

```
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
```

```
import threading
```

```
import io
```

```
import pybase64
```

```
import torch
```

```
from vllm.utils.sparse_utils import check_sparse_tensor_invariants_threadsafe
```

```
# 辅助函数 _encode_tensor, _create_valid_dense_tensor, _create_malicious_sparse_tensor 省略
```

```
class TestConcurrentRaceProtection:
```

```
    """验证锁能防止 A-enter, B-enter, A-exit, B-load 竞态。"""
```

```
    def test_forced_interleaving_still_rejects_invalid(self, model_config):
```

```
        """强制构造无锁时能绕过检查的交错执行序列。
```

线程 A（良性）：进入上下文 -> 加载有效张量 -> 退出上下文。

线程 B（恶意）：在 A 进入后尝试进入，但只有在 A 退出后才执行加载。

如果没有 `\_SPARSE\_LOAD\_LOCK`，当 B 加载时全局标志已被设为 False，从而绕过检查。有了锁后，B 必须等到 A 完全退出才能进入。

```
    """
```

```
    barrier = threading.Barrier(2, timeout=5)
```

```
    results = {}
```

```
    valid_encoded = _encode_tensor(_create_valid_dense_tensor())
```

```
    malicious_encoded = _encode_tensor(_create_malicious_sparse_tensor())
```

```
    def thread_a_benign():
```

```
        try:
```

```
            with check_sparse_tensor_invariants_threadsafe():
```

```
                barrier.wait() # 通知 B 当前线程已持有锁
```

```
                tensor = torch.load(
```

```
                    io.BytesIO(pybase64.b64decode(valid_encoded, validate=True)),
```

```
                    weights_only=True,
```

```
                    map_location=torch.device("cpu"),
```

```
                )
```

```
                results["a_loaded"] = True
```

```
            except threading.BrokenBarrierError:
```

```
                results["a_barrier_broken"] = True
```

```
    def thread_b_malicious():
```

```

try:
    barrier.wait() # 等待 A 进入锁后
    with check_sparse_tensor_invariants_threadsafe():
        tensor = torch.load(
            io.BytesIO(pybase64.b64decode(malicious_encoded, validate=True)),
            weights_only=True,
        )
        tensor.to_dense()
        results["b_bypass"] = True
except (RuntimeError, ValueError):
    results["b_rejected"] = True
except threading.BrokenBarrierError:
    results["b_barrier_broken"] = True

t_a = threading.Thread(target=thread_a_benign)
t_b = threading.Thread(target=thread_b_malicious)
t_a.start()
t_b.start()
t_a.join()
t_b.join()

# 断言良性线程成功加载，恶意线程被拒绝
assert results.get("a_loaded"), "良性线程应成功加载"
assert results.get("b_rejected"), "恶意线程应被拒绝"

```

评论区精华

- DarkLight1337 建议将锁和原始上下文管理器封装成一个独立的管理器，而不是让调用者手动组合。作者按建议重构，后续重命名为 `check_sparse_tensor_invariants_threadsafe`。
- DarkLight1337 要求函数名从 `sparse_invariants_checked` 改为现名，以更清晰表达线程安全语义。作者已修改。
- DarkLight1337 认为上下文管理器上的文档字符串冗余，要求移除或切换为 `mkdocs` 风格（单反引号）。作者移除了上下文管理器上的文档字符串，并调整了模块文档字符串。
- DarkLight1337 要求测试文件同样处理文档字符串，作者移除了测试文件的文档字符串。
- 包装为统一上下文管理器 (design): 作者按建议实现，后续重命名为 `check_sparse_tensor_invariants_threadsafe`。
- 重命名函数 (style): 作者已重命名。
- 文档字符串冗余和风格 (documentation): 作者移除了上下文管理器上的文档字符串，并调整模块文档字符串为 `mkdocs` 风格。
- 测试文件文档字符串 (documentation): 作者移除了测试文件的文档字符串。

风险与影响

- 风险:

- 性能：全局锁可能带来微小争用，但嵌入加载路径并非高频，锁持有时间极短（仅标志操作），影响可接受。
- 兼容性：新包装器完全替代旧行为，所有调用者自动获得线程安全，无 API 变更。
- 安全：彻底封堵了因并发绕过安全检测的路径，降低了 CVE-2025-62164 被利用的风险。
- 影响：
 - 用户：无功能可见变化，但安全性提升。
 - 系统：所有多模态嵌入（音频、图像、提示嵌入）加载路径自动获得保护。
 - 团队：新增一个约 40 行的实用模块，维护成本极低。
 - 风险标记：并发竞争修复，核心安全路径

关联脉络

- 暂无明显关联 PR