PR #49948 完整报告

vllm-project/vllm

Fix DoS via sample-rate forgery bypassing audio decode duration guard

合并时间：2026-08-10 23:49

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/49948

执行摘要

- 一句话：音频解码新增字节上限，封堵采样率伪造 OOM DoS
- 推荐动作：值得精读。这是一个小而完整的安全修复样本：问题定位清晰（时长守卫信任容器头）、修复设计巧妙（引入与采样率解耦的独立字节守卫，两道防线互补）、测试覆盖到位（直接构造 PoC 场景验证），且 review 闭环质量高（导入整理、去掉测试类、环境变量缓存、常量复用等意见均被采纳）。建议关注 load_audio_soundfile 的预读估算与 load_audio_pyav 的增量累计这两条路径的差异，以及后续同类资源守卫（图片、视频）如何复用同一模式。

功能与动机

PR body 明确指出现有 max_duration_s 守卫的可利用性："An attacker can set samplerate=655350 (FLAC max) with 8 channels to make hundreds of millions of frames appear as a short clip, bypassing the duration check while f.read() allocates up to 11.7 GiB of float32 PCM — enough to OOM-kill the API server." 修复目标是在时长守卫之外增加一个独立于采样率的字节预算兜底："Both guards (duration + bytes) are independently useful: duration catches legitimately long files early; bytes catches sample-rate forgery regardless of claimed duration." 社区成员 TobyB1702 在 issue 评论中实测复现了漏洞（31 MiB PCM 分配被新守卫精确拦截），并指出了 security.md 缺少该环境变量的文档缺口。

实现拆解

实现分五步完成：

1. 注册环境变量（vllm/envs.py）：在 VLLMEnvironment 数据类中新增字段 VLLM_MAX_AUDIO_DECODE_BYTES: int = 268_435_456（256 MiB），在 environment_variables 字典中注册解析 lambda 和设计注释（说明默认值可覆盖 600 秒单声道 48 kHz float32），并加入 compile_factors 的变量列表，保证多进程 worker 通过环境变量传播。
2. 实现字节守卫核心（vllm/multimodal/media/audio.py）：给 load_audio_soundfile 和 load_audio_pyav 增加 max_decode_bytes 参数。soundfile 路径在 f.read() 之前用 f.frames * f.channels * np.dtype(np.float32).itemsize 估算 PCM 大小并提前拒绝；PyAV 路径无法预知帧数，因此在解码循环中用 total_decode_bytes += arr.nbytes 增量累计，每帧解码后检查阈值。load_audio 的两个回退分支均透传该参数，

MediaIO.load_bytes / load_file 则直接从 envs.VLLM_MAX_AUDIO_DECODE_BYTES 取值。

3. STT 服务接入（vllm/entrypoints/speech_to_text/base/serving.py）：在服务 __init__ 中缓存 self.max_audio_decode_bytes = envs.VLLM_MAX_AUDIO_DECODE_BYTES（按 review 意见避免每次请求重复求值环境 lambda），并在 _decode_and_chunk_speech 中传给 load_audio。
4. 回归测试（tests/multimodal/media/test_audio.py）：新增 _make_flac_bytes 内联构造最小 FLAC 文件，覆盖四个场景：小文件同时通过时长与字节守卫、frames * channels * 4 超过限制时拒绝、PoC 伪造采样率场景（655350 Hz、8 声道、100 万帧，时长守卫误判为 1.5 秒但字节守卫拒绝）、以及 load_audio 向后端透传 max_decode_bytes。
5. 文档配套（docs/usage/security.md）：在安全配置表中新增 VLLM_MAX_AUDIO_DECODE_BYTES 条目，说明其防采样率伪造的作用，并保留 " 设为 0 禁用限制 " 的警告文案。

关键文件：

- vllm/multimodal/media/audio.py（模块 音频解码；类别 source；类型 core-logic；符号 load_audio_soundfile, load_audio_pyav, load_audio, AudioMediaIO.load_bytes）：修复核心所在：为 soundfile 与 PyAV 两条解码路径分别增加与采样率解耦的字节预算守卫，并让 load_audio 与 MediaIO 透传新参数。
- tests/multimodal/media/test_audio.py（模块 音频测试；类别 test；类型 test-coverage；符号 _make_flac_bytes, test_small_file_passes_memory_guard, test_memory_guard_rejects_large_allocation, test_forged_samplerate_rejected_by_memory_guard）：新增 4 个回归测试并直接构造 PoC 场景（伪造采样率 655350 Hz、8 声道），验证字节守卫在时长守卫被绕过时仍能拒绝分配。
- vllm/envs.py（模块 环境变量；类别 source；类型 configuration；符号 VLLM_MAX_AUDIO_DECODE_BYTES）：定义并注册 VLLM_MAX_AUDIO_DECODE_BYTES 环境变量，默认 256 MiB，并将该变量纳入多进程环境传播列表。
- vllm/entrypoints/speech_to_text/base/serving.py（模块 语音服务；类别 source；类型 core-logic；符号 init, _decode_and_chunk_speech）：STT 服务入口将新环境变量缓存并在每次音频解码时传入 load_audio，是修复在线上生效的接入点。
- docs/usage/security.md（模块 安全文档；类别 docs；类型 documentation）：补充安全配置表，记录新环境变量的默认值与防采样率伪造用途，方便部署方按需调整。

关键符号：load_audio_soundfile, load_audio_pyav, load_audio, AudioMediaIO.load_bytes, AudioMediaIO.load_file, _decode_and_chunk_speech, _make_flac_bytes, test_forged_samplerate_rejected_by_memory_guard

关键源码片段

vllm/multimodal/media/audio.py

修复核心所在：为 soundfile 与 PyAV 两条解码路径分别增加与采样率解耦的字节预算守卫，并让 load_audio 与 MediaIO 透传新参数。

```
# 关键修复单元 1：soundfile 后端预读字节守卫
# 原理：在 f.read() 真正分配内存之前，用容器头部的 frames * channels * 4
# 估算 float32 PCM 大小。该估算不依赖采样率，因此伪造高采样率无法绕过。
def load_audio_soundfile(
    path: BytesIO | Path | str,
    *,
    sr: float | None = 22050,
    mono: bool = True,
    max_duration_s: float | None = None,
    max_decode_bytes: int | None = None,
) -> tuple[np.ndarray, int]:
    """加载音频（soundfile 后端）"""
    with soundfile.SoundFile(path) as f:
        native_sr = f.samplerate
        # 时长守卫：f.frames / native_sr，信任容器头，可被伪造采样率绕过
        if max_duration_s is not None:
            file_duration_s = f.frames / native_sr
            if file_duration_s > max_duration_s:
                raise ValueError(
                    f"Audio exceeds maximum allowed duration of "
                    f"{max_duration_s}s (file contains "
                    f"{file_duration_s:.1f}s at {native_sr}Hz). Set "
                    f"VLLM_MAX_AUDIO_DECODE_DURATION_S to "
                    f"increase this limit."
                )

        # 字节守卫：独立于采样率，按 frames * channels * 4B 提前估算
        if max_decode_bytes is not None:
            estimated_bytes = (
                f.frames * f.channels * np.dtype(np.float32).itemsize
            )
            if estimated_bytes > max_decode_bytes:
                raise ValueError(
                    f"Audio would allocate {estimated_bytes / MiB_bytes:.0f} "
                    f"MiB of PCM ({f.frames} frames x {f.channels} channels"
                    f" x 4B), exceeding the "
                    f"{max_decode_bytes / MiB_bytes:.0f} MiB limit. Set "
                    f"VLLM_MAX_AUDIO_DECODE_BYTES to increase this limit."
                )

        # 通过检查后才真正读取，恶意文件不会触发大块内存分配
        y = f.read(dtype="float32", always_2d=False).T

    if mono and y.ndim > 1:
        y = np.mean(y, axis=tuple(range(y.ndim - 1)))

    if sr is not None and sr != native_sr:
```

```
            y = resample_audio_pyav(y, orig_sr=native_sr, target_sr=sr)
            return y, int(sr)
        return y, native_sr

    # 关键修复单元 2：PyAV 后端的解码中增量字节计数
    # 原理：PyAV 无法预知帧数，因此在 decode 循环里用 arr.nbytes 累计实际
    # 解码产生的字节数，每帧解码后检查是否超过 max_decode_bytes。
    # 累计的是中间解码数组（含重采样结果），口径偏保守但安全侧有利。
        total_samples = 0
        total_decode_bytes = 0 # 字节预算累计器，不信任容器头部信息

        for frame in container.decode(stream):
            if needs_resampling:
                assert resampler is not None
                for out_frame in resampler.resample(frame):
                    arr = out_frame.to_ndarray()
                    total_samples += arr.shape[-1]
                    total_decode_bytes += arr.nbytes
                    chunks.append(arr)
            else:
                arr = frame.to_ndarray()
                total_samples += arr.shape[-1]
                total_decode_bytes += arr.nbytes
                chunks.append(arr)

            # 时长守卫：按目标采样率折算的样本数上限，检测合法长音频
            if max_samples is not None and total_samples > max_samples:
                raise ValueError(
                    f"Audio exceeds maximum allowed duration of "
                    f"{max_duration_s}s (decoded {total_samples} "
                    f"samples at {sr}Hz). Set "
                    f"VLLM_MAX_AUDIO_DECODE_DURATION_S to increase this limit."
                )

            # 字节守卫：采样率伪造场景下时长守卫可能失效，用字节数兜底
            if (
                max_decode_bytes is not None
                and total_decode_bytes > max_decode_bytes
            ):
                raise ValueError(
                    f"Audio decode exceeded "
                    f"{max_decode_bytes / MiB_bytes:.0f} MiB memory "
                    f"limit ({total_decode_bytes / MiB_bytes:.0f} MiB "
                    f"decoded so far). Set "
                    f"VLLM_MAX_AUDIO_DECODE_BYTES to increase this limit."
                )
```

tests/multimodal/media/test_audio.py

新增 4 个回归测试并直接构造 PoC 场景（伪造采样率 655350 Hz、8 声道），验证字节守卫在时长守卫被绕过时仍能拒绝分配。

```
def _make_flac_bytes(frames: int, channels: int, samplerate: int) -> bytes:
    """在内存中构造一个最小 FLAC 文件，避免依赖真实音频资源。"""
    data = np.zeros((frames, channels), dtype=np.int16)
    buf = BytesIO()
    sf.write(buf, data, samplerate, format="FLAC")
    return buf.getvalue()


def test_forged_samplerate_rejected_by_memory_guard():
    """核心 PoC：高采样率绕过时长守卫，但字节守卫必须兜底拒绝。"""
    # 伪造采样率 655350 Hz（FLAC 上限）+ 8 声道 + 100 万帧：
    # 时长守卫看到 1_000_000 / 655_350 ≈ 1.5 s，会放行；
    # 而真实 PCM 分配需要 1_000_000 * 8 * 4 = 32 MB，超过 16 MiB 限制。
    payload = _make_flac_bytes(frames=1_000_000, channels=8, samplerate=655350)
    with pytest.raises(ValueError, match="VLLM_MAX_AUDIO_DECODE_BYTES"):
        load_audio_soundfile(
            BytesIO(payload),
            sr=None,
            max_duration_s=600,
            max_decode_bytes=16 * 1024 * 1024,
        )


def test_load_audio_threads_max_decode_bytes():
    """验证 load_audio 能把 max_decode_bytes 透传到后端解码器。"""
    # 50_000 帧 * 4 声道 * 4 字节 = 800 KB，限制 512 KB 应被拒绝
    payload = _make_flac_bytes(frames=50_000, channels=4, samplerate=44100)
    with pytest.raises(ValueError, match="VLLM_MAX_AUDIO_DECODE_BYTES"):
        load_audio(
            BytesIO(payload),
            sr=None,
            max_duration_s=600,
            max_decode_bytes=512 * 1024,
        )
```

评论区精华

评审中存在四轮有实质内容的讨论，全部已解决：

- 测试导入整理：DarkLight1337 要求 "Please clean up the PR by moving imports to top level"，作者随后将 io、soundfile 等导入提升到模块顶层。
- 测试类结构去留：NickLucche 对 TestMaxDecodeBytes 分组提出意见："can we drop the test class? I am not a fan of the grouping pattern claude uses and class is just overhead here"，最终作者拆掉了测试类，改为顶层辅助函数加独立测试函数，代码更简洁，也与仓库现有测试风格一致。

- 环境变量缓存：NickLucche 指出 envs.VLLM_MAX_AUDIO_DECODE_BYTES 是运行时求值的 lambda，"probably worth to just cache it once in self in init"，作者在 serving.py 的 __init__ 中缓存到 self.max_audio_decode_bytes。
- 常量复用：DarkLight1337 建议错误信息中的 MiB 换算改用 MiB_bytes 常量而非手写 1024**2，已按意见修改。

此外 TobyB1702 在 issue 评论中完整复现了漏洞与新守卫的拦截行为，并推动 security.md 补文档。

- 测试导入整理到模块顶层 (style): 作者回复 "Pushed" 后，BytesIO、soundfile 等导入被提升到文件顶部，与仓库 pytest 测试风格保持一致。
- 测试类分组模式是否多余 (design): 最终 head 版本中测试类被移除，改为顶层辅助函数 _make_flac_bytes 加独立测试函数，验收了该设计意见。
- 环境变量求值应在 __init__缓存 (performance): 作者确认 "This should also be solved"，在 serving.py 的 __init__ 中缓存为 self.max_audio_decode_bytes。
- MiB 换算复用 MiB_bytes 常量 (style): 已修改为 from vllm.utils.mem_constants import MiB_bytes，错误信息统一用该常量换算。
- VLLM_MAX_AUDIO_DECODE_BYTES 未写入 security.md (documentation): 作者回应已将该参数加入 docs/usage/security.md 的安全配置表，并说明安全通告因尚未发布而未链接到 PR。

风险与影响

- 风险：主要风险点如下：
- 合法音频回归：默认 256 MiB 上限对 600 秒单声道 48 kHz float32（约 115 MB）是足够的，但多声道或更高采样率的合法长音频可能被新限制拒绝。文档已说明可通过 VLLM_MAX_AUDIO_DECODE_BYTES 调高，且设 0 可禁用（不推荐）。
- PyAV 计数口径偏保守：load_audio_pyav 的 total_decode_bytes 按解码帧 arr.nbytes 累计，重采样路径统计的是中间 fltp 数组而非最终 float32 mono 数组，可能提前触发阈值。这偏向安全侧，但调用方需理解该语义差异。
- 环境变量传播：VLLM_MAX_AUDIO_DECODE_BYTES 已加入 compile_factors 列表，多进程场景下可正常传播；若部署环境遗漏，默认值仍生效。
- 守卫覆盖范围：字节守卫只覆盖 AudioMediaIO 与 STT 服务入口，其他直接调用 load_audio_pyav 的路径若不显式传 max_decode_bytes 则无保护，需团队后续排查是否还有未接入的调用点。
- 影响：影响范围集中在多模态音频输入链路：所有通过 OpenAI STT 服务或 AudioMediaIO 加载音频的用户默认获得 256 MiB 的 PCM 解码上限，攻击者无法再通过伪造容器头采样率触发大规模内存分配。对正常单声道语音文件（Whisper 等模型常用 16 kHz）影响可忽略；对多声道高采样率素材，部署方需按需调高环境变量。从团队视角，该 PR 与既有的 VLLM_MAX_IMAGE_PIXELS、VLLM_MAX_AUDIO_CLIP_FILESIZE_MB、VLLM_MAX_AUDIO_DECODE_DURATION_S 共同构成多模态输入的 " 解码预算 " 防线，完善了安全配置矩阵，也为此类漏洞提供了可复用的纵深防御模式。

- 风险标记：安全修复（DoS），默认限制 256 MiB，跨 soundfile/PyAV 双后端，独立字节守卫兜底，文档已同步

关联脉络

- PR #51657 [2/N] Harden Transformers modelling backend multi-modal path: 与本次修复同属维护者对多模态输入路径的安全加固系列：该 PR 加固 Transformers 后端多模态权重加载与跳过逻辑，本 PR 则加固音频解码的资源边界（PCM 字节上限），两者共同降低多模态入口被恶意输入攻击的风险。