

PR #51445 完整报告

vllm-project/vllm

[Frontend] Use server-generated keys for late-interaction query caches

合并时间: 2026-09-03 15:15

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/51445>

执行摘要

- 一句话: 修复 late-interaction 评分缓存键使用客户端可控请求 ID 导致的缓存污染安全漏洞。
- 推荐动作: 值得精读。这是一个典型的安全漏洞修复, 其设计模式 (引入不可猜测的服务端密钥并管控其作用域) 具有通用借鉴价值。实现简洁、直接, 附带了针对性的回归测试, 是高质量 bugfix 的范例。建议安全相关模块的开发者重点关注其对上下文管理和阶段间状态传递的处理方式。

功能与动机

根据 PR body 描述, 这是一个公开的安全修复, 源于安全公告 GHSA-2phq-3phc-84px。其动机是防止攻击者通过猜测或知道其他客户端的 `X-Request-Id` 并赢得并发竞争, 来改变受害者的 `/score` 或 `/rerank` 结果。漏洞详情为: “A network client who knows or guesses another client's request ID and wins a concurrency race could change that client's `/score` or `/rerank` result.”

实现拆解

1. 扩展上下文数据结构: 在 `vllm/entrypoints/pooling/typing.py` 的 `PoolingServeContext` 数据类型中新增 `late_interaction_query_keys: list[str] | None = None` 字段。该字段作为在请求生命周期内 (查询编码到文档编码) 安全传递服务端生成密钥的容器。
2. 修改查询编码逻辑: 在 `vllm/entrypoints/pooling/scoring/serving.py` 的 `_flash_late_interaction_encode_queries` 方法中, 引入 `from vllm.utils import random_uuid`。将原先基于 `ctx.request_id` 构建的键 (`query_keys = [f"{ctx.request_id}-query-{}"]`) 替换为基于随机 UUID 的键 (`query_namespace = random_uuid()` 然后 `f"late-interaction-{query_namespace}-query-{}"`)。新生成的键列表被存储到 `ctx.late_interaction_query_keys`。此变更消除了客户端对键命名空间的控制能力。
3. 同步文档编码逻辑: 在 `_flash_late_interaction_encode_docs` 方法中, 不再独立生成 `query_keys`, 而是从上下文读取 `ctx.late_interaction_query_keys`。增加防御性检查, 若为 `None` 则抛出 `RuntimeError`, 明确指示初始化缺失。这确保文档阶段使用与查询阶段完全相同的服务端生成的密钥。
4. 添加回归测试: 新增测试文件 `tests/entrypoints/pooling/scoring/test_late_interaction_serving.py`。测试用例 `test_colliding_request_ids_use_distinct_query_cache_keys` 创建两个共享相同 `request_id` 的上下文, 执行完整的查询 / 文档编码流程, 并断言它们生成的查询键和文档键各自相同, 但两个请求间的键不同, 且键中不包含原始的调用方可控 ID。此测

试直接验证了修复的有效性。

关键文件：

- `vllm/entrypoints/pooling/scoring/serving.py`（模块 评分服务；类别 `source`；类型 `core-logic`；符号 `_flash_late_interaction_encode_queries`, `_flash_late_interaction_encode_docs`）：核心修复逻辑所在，修改了查询键生成和重用方式，是安全漏洞修复的直接实现点。
- `vllm/entrypoints/pooling/typing.py`（模块 上下文类型；类别 `source`；类型 `data-contract`；符号 `PoolingServeContext`）：为共享状态引入新的数据结构字段 `late_interaction_query_keys`，是实现查询与文档阶段间密钥传递的关键支撑。
- `tests/entrypoints/pooling/scoring/test_late_interaction_serving.py`（模块 评分测试；类别 `test`；类型 `test-coverage`；符号 `_make_context`, `_query_key`, `test_colliding_request_ids_use_distinct_query_cache_keys`）：新增回归测试，直接验证修复效果：两个共享相同客户端请求 ID 的上下文必须生成不同的缓存键。

关键符号：`_flash_late_interaction_encode_queries`, `_flash_late_interaction_encode_docs`, `test_colliding_request_ids_use_distinct_query_cache_keys`

关键源码片段

`vllm/entrypoints/pooling/scoring/serving.py`

核心修复逻辑所在，修改了查询键生成和重用方式，是安全漏洞修复的直接实现点。

```
# 在查询编码阶段生成并存储服务端控制的缓存键命名空间
async def _flash_late_interaction_encode_queries(self, ctx: ScoringServeContext):
    # ... 前置断言和输入准备 ...
    query_namespace = random_uuid() # 生成不可预测的、服务端控制的 UUID
    query_keys = [
        f"late-interaction-{query_namespace}-query-{i}" for i in range(n_queries)
    ]
    ctx.late_interaction_query_keys = query_keys # 存储到上下文，供后续文档阶段使用
    # ... 后续设置 pooling_params 和调用引擎 ...

# 在文档编码阶段，从上下文读取并重用查询键，确保阶段间一致性
async def _flash_late_interaction_encode_docs(self, ctx: ScoringServeContext):
    # ... 前置断言和输入准备 ...
    query_keys = ctx.late_interaction_query_keys
    if query_keys is None:
        raise RuntimeError("Late-interaction query keys were not initialized.") # 防御性检查
    # ... 后续使用 query_keys 为文档设置 pooling_params ...
```

`tests/entrypoints/pooling/scoring/test_late_interaction_serving.py`

新增回归测试，直接验证修复效果：两个共享相同客户端请求 ID 的上下文必须生成不同的缓存键。

```
import pytest
```

```

from unittest.mock import AsyncMock, Mock
from vllm.entrypoints.pooling.scoring.serving import ServingScores
from vllm.entrypoints.pooling.typing import PoolingServeContext
from vllm.inputs import tokens_input
from vllm.pooling_params import PoolingParams

def _make_context() -> PoolingServeContext:
    """创建一个共享相同 request_id 的模拟上下文。"""
    return PoolingServeContext(
        request=Mock(),
        model_name="model",
        request_id="score-caller-controlled-id", # 模拟客户端可控的 ID
        pooling_params=PoolingParams(task="token_embed"),
        lora_request=None, priorities=None, prompt_extras=None,
        engine_inputs=[
            tokens_input(prompt_token_ids=[1]),
            tokens_input(prompt_token_ids=[2]),
        ],
        n_queries=1,
    )

def _query_key(context: PoolingServeContext) -> str:
    """从引擎输入参数中提取查询键。"""
    pooling_params = context.engine_inputs[0]["params"]
    late_interaction_params = pooling_params.late_interaction_params
    assert late_interaction_params is not None
    return late_interaction_params.query_key

@pytest.mark.asyncio
async def test_colliding_request_ids_use_distinct_query_cache_keys():
    """验证修复：两个具有相同客户端 ID 的请求必须获得不同的服务端缓存键。"""
    serving = object.__new__(ServingScores)
    serving._prepare_generators = AsyncMock()
    serving._collect_batch = AsyncMock()

    first = _make_context()
    second = _make_context()
    for context in (first, second):
        await serving._flash_late_interaction_encode_queries(context)
        await serving._flash_late_interaction_encode_docs(context)

    # 获取每次 _prepare_generators 调用时传入的上下文
    prepared_contexts = [
        call.args[0] for call in serving._prepare_generators.await_args_list
    ]

    # 提取键：第 0 次调用是第一个请求的查询，第 1 次是第一个请求的文档，以此类推
    first_query_key = _query_key(prepared_contexts[0])
    first_doc_key = _query_key(prepared_contexts[1])

```

```
second_query_key = _query_key(prepared_contexts[2])
second_doc_key = _query_key(prepared_contexts[3])

# 断言每个请求内部查询和文档键一致，但两个请求间的键不同
assert first_query_key == first_doc_key
assert second_query_key == second_doc_key
assert first_query_key != second_query_key
# 关键安全断言：生成的键中不能包含客户端可控的 ID 字符串
assert "caller-controlled-id" not in first_query_key
assert "caller-controlled-id" not in second_query_key
```

评论区精华

本 PR 的 review 过程较为直接。维护者 [noooop](#) 直接批准了 PR 并评论“thanks!”。自动化工具 [claude\[bot\]](#) 因 PR 来自 fork 而未进行自动审查。没有发现关于设计或实现细节的争议性讨论，修复方案清晰且直接针对安全公告。

- 安全漏洞修复方案审查与合并 (security): 采用服务端生成随机 UUID 作为缓存键命名空间的方案被接受，并合入主分支。

风险与影响

- 风险：技术风险：
 - 回归风险：低。变更逻辑清晰，从“重建键”改为“存储并重用键”，核心的缓存查找逻辑未变。新增的防御性检查（if query_keys is None: raise）提供了明确的失败路径，而非静默错误。
 - 兼容性风险：中低。在 PoolingServeContext 中新增了 late_interaction_query_keys 可选字段，任何反序列化或直接构造该上下文的代码可能需要适配，但因其有默认值 None，向后兼容性较好。
 - 并发风险：修复本身旨在解决并发安全问题。随机 UUID 的引入确保了并发请求的隔离性。

主要风险点：在于此修复假设所有调用 `_flash_late_interaction_encode_docs` 的路径都必须经过 `_flash_late_interaction_encode_queries`。如果存在直接调用 `_flash_late_interaction_encode_docs` 的路径，可能触发 `RuntimeError`。

- 影响：直接影响：增强了使用 Flash late-interaction 评分（`/score` 或 `/rerank`）端点的安全性，防止了通过请求 ID 猜测进行的缓存污染攻击。对终端用户而言，修复后其评分结果不再可能被其他恶意客户端并发篡改。系统影响：修复集中在 `vllm.entrypoints.pooling` 模块内的评分服务逻辑，影响范围有限。安全修复提升了整个评分服务的可信度。团队影响：为处理类似安全公告（GHSA）提供了清晰的修复范式：使用服务端可控的标识符替代客户端提供的标识符，并在关键阶段间共享该标识符。测试用例展示了如何验证此类安全修复。
- 风险标记：防御性检查触发路径，上下文数据结构变更

关联脉络

- 暂无明显关联 PR