PR #52523 完整报告

vllm-project/vllm

[Bugfix] Redact api_key in startup logs and compile cache factors

合并时间：2026-08-19 15:52

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/52523

执行摘要

- 一句话：脱敏启动日志与编译缓存中的 api_key，修复密钥泄漏
- 推荐动作：值得精读，属于低成本高收益的安全修复范例。值得关注的设计决策：脱敏只挂在日志路径上，刻意不动 get_non_default_args() 与 --args-json，避免影响 Rust 前端数据流；_redact_sensitive_args 在无敏感字段时返回原对象，减少无谓拷贝；测试用 caplog 断言日志最终文本，直接锁定泄漏面。可借鉴的改进方向：把敏感字段定义下沉到参数元数据（如 argparse 的 sensitive 标记），或扩展为模式脱敏，减少硬编码白名单的维护成本。

功能与动机

PR body 明确指出两条泄漏路径：API server 启动日志 'non-default args: ...' 会原样打印所有非默认 CLI 参数（包括 --api-key 与 VLLM_API_KEY），任何有日志读取权限的人（docker logs、主机文件系统、log shippers）都能从单行启动日志恢复密钥；同时 compile_factors() 从所有注册的 VLLM_* 环境变量出发构造缓存键，并把原始因子字典持久化到 cache_key_factors.json，VLLM_API_KEY 未在忽略列表中，密钥被明文写盘且每次轮换密钥都会使编译缓存失效。作者强调 'The API key was only ever logged, never sent anywhere — so the leak is free real estate for anyone reading logs'，属于低成本高收益的安全修复。

实现拆解

变更入口在 vllm/entrypoints/serve/utils/api_utils.py 的日志函数与 vllm/envs.py 的编译缓存因子枚举，配套测试覆盖两个泄漏面。

1. 日志脱敏（vllm/entrypoints/serve/utils/api_utils.py）：新增模块级常量 _SENSITIVE_ARG_FIELDS = frozenset({"api_key"}) 与函数 _redact_sensitive_args(args)。该函数先检查入参中是否含敏感字段，若无则原样返回（避免多余拷贝），有则将对应字段值替换为 "***"。log_non_default_args() 改用脱敏后的字典输出；get_non_default_args() 与 --args-json 传递路径不受影响，Rust 前端数据流保持不变。
2. 编译缓存因子排除（vllm/envs.py）：在 compile_factors() 的 ignored_factors 集合中新增 "VLLM_API_KEY"，并附注释说明其为凭据、不影响编译产物且不得持久化到 cache_key_factors.json。此后该变量既不参与 torch.compile 缓存键哈希，也不会被 backends.py 写入缓存元数据文件；密钥轮换不再导致编译缓存失效。

3. 测试配套：tests/entrypoints/serve/utils/test_api_utils.py 新增 TestRedactSensitiveArgs 三个用例，分别验证「仅替换敏感值且不改原字典」「无敏感字段时返回原对象」「经 caplog 断言日志无明文密钥但保留 'api_key': '***' 且非敏感参数仍在」；tests/test_envs.py 新增 test_api_key_is_not_compile_factor 验证 compile_factors() 排除 VLLM_API_KEY。
4. 验证与收尾：作者本地运行 ruff 与 pytest（14 passed，含 3 个新脱敏测试）；CI 经历 pre-commit 格式修复与多次 Buildkite 触发后，由 DarkLight1337 approve 并合并。

关键文件：

- vllm/entrypoints/serve/utils/api_utils.py（模块 参数工具；类别 source；类型 entrypoint；符号 _redact_sensitive_args, log_non_default_args, _SENSITIVE_ARG_FIELDS）：日志脱敏的核心实现，新增 _redact_sensitive_args 并在 log_non_default_args 中接入，是修复的第一泄漏面。
- vllm/envs.py（模块 环境变量；类别 source；类型 core-logic；符号 compile_factors）：compile_factors() 的 ignored_factors 新增 VLLM_API_KEY，堵住第二个泄漏面（缓存元数据写盘与缓存键哈希）。
- tests/entrypoints/serve/utils/test_api_utils.py（模块 参数工具；类别 test；类型 test-coverage；符号 TestRedactSensitiveArgs, test_redact_replaces_sensitive_values_only, test_no_sensitive_fields_returns_original, test_api_key_not_in_log）：新增 TestRedactSensitiveArgs 三个用例，用 caplog 直接断言日志输出中不出现明文密钥，验证脱敏行为与边界条件。
- tests/test_envs.py（模块 环境变量；类别 test；类型 test-coverage；符号 test_api_key_is_not_compile_factor）：新增 test_api_key_is_not_compile_factor，防止 VLLM_API_KEY 从 ignored_factors 回归泄漏。

关键符号：_redact_sensitive_args, log_non_default_args, compile_factors

关键源码片段

vllm/entrypoints/serve/utils/api_utils.py

日志脱敏的核心实现，新增 _redact_sensitive_args 并在 log_non_default_args 中接入，是修复的第一泄漏面。

```
# 敏感参数白名单：只有这些字段的值不允许原样出现在日志中。
# 新增敏感参数时需要同步扩展该集合。
_SENSITIVE_ARG_FIELDS = frozenset({"api_key"})


def _redact_sensitive_args(args: dict[str, Any]) -> dict[str, Any]:
    """返回一份敏感值被脱敏的 args 副本，仅供日志输出使用。

    若 args 中不含敏感字段，直接返回原对象，避免无意义的拷贝；
    含敏感字段时新建字典，并把对应字段值替换为 "***"。
    """
    if not any(key in _SENSITIVE_ARG_FIELDS for key in args):
        return args
```

```
    return {
        key: ("***" if key in _SENSITIVE_ARG_FIELDS else value)
        for key, value in args.items()
    }


def log_non_default_args(args: Namespace | EngineArgs):
    non_default_args = get_non_default_args(args)
    # 日志中保留字段名、值统一脱敏，操作员仍可判断是否配置了 api_key，
    # 但无法再通过日志恢复密钥明文。
    logger.info("non-default args: %s", _redact_sensitive_args(non_default_args))
```

vllm/envs.py

compile_factors() 的 ignored_factors 新增 VLLM_API_KEY，堵住第二个泄漏面（缓存元数据写盘与缓存键哈希）。

```
    # compile_factors() 会遍历所有已注册的 vLLM 环境变量，把未被忽略的
    # 变量哈希进 torch.compile 缓存键；backends.py 还会把原始因子字典
    # 持久化到 cache_key_factors.json。因此所有凭据必须加入 ignored_factors。
    ignored_factors: set[str] = {
        # 网络 / 路径类变量：不参与编译产物，也避免搬家导致缓存失效。
        "VLLM_RPC_BASE_PATH",
        "VLLM_HOST_IP",
        "LD_LIBRARY_PATH",
        # 持久化路径：只影响缓存 / 配置目录位置，不影响编译产物。
        "VLLM_CACHE_ROOT",
        "VLLM_CONFIG_ROOT",
        "VLLM_FLASHINFER_AUTOTUNE_CACHE_DIR",
        # 凭据类：S3 与 API key。绝不能以明文形式写入 cache_key_factors.json。
        "S3_ACCESS_KEY_ID",
        "S3_SECRET_ACCESS_KEY",
        "S3_ENDPOINT_URL",
        # 新增：从编译缓存因子中排除 VLLM_API_KEY，避免密钥写盘与缓存键漂移。
        "VLLM_API_KEY",
        "VLLM_USAGE_STATS_SERVER",
        ...
    }
```

评论区精华

核心讨论来自 vrdn-23 的补充建议与作者回应：vrdn-23 指出 'Also probably need to fix this in the rust backend and as a completeness measure, make sure it is in the IGNORE_LIST in vllm/envs.py, so that it doesn't get written to the disk for the compile cache.' 作者回应已完成 compile_factors() 排除并新增回归测试，同时说明 Rust 后端 api_keys 字段已使用脱敏 Debug 实现（rust/src/server/src/config.rs 的 fmt_redacted_api_keys，cli/tests.rs 断言 api_keys: [; 2]），无需改动。此外 mergify[bot] 提示 pre-commit 失败，作者以 ruff v0.14.0 格式化提交解决；claude[bot] 说明 fork PR 默认不自动 review。

- 补全泄漏面：Rust 后端与编译缓存忽略列表 (security): 作者将 VLLM_API_KEY 加入 compile_factors() 的 ignored_factors 并新增回归测试；Rust 侧确认已通过 fmt_redacted_api_keys 脱敏 Debug 输出覆盖，无需改动。
- pre-commit 格式检查失败 (style): 作者在第二个提交中应用 ruff v0.14.0 格式化（Co-authored-by: Claude），后续 CI 通过。
- fork PR 自动 review 关闭 (other): 未触发额外 bot review，由 DarkLight1337 人工 approve。

风险与影响

- 风险：
  1. 脱敏范围依赖硬编码白名单：_SENSITIVE_ARG_FIELDS 只含 api_key，未来新增凭据类参数（如自定义认证头）需要手动补名单，存在漏配风险。
  2. 脱敏仅作用于顶层键：若敏感值出现在嵌套 dict 中（当前 get_non_default_args 输出为扁平结构）将不会被覆盖。
  3. 覆盖路径有限：仅 log_non_default_args 一条日志路径做了脱敏，异常堆栈或其它调试日志仍可能打印含密钥的完整参数对象。
  4. 编译缓存键语义变化：VLLM_API_KEY 不再进入缓存键，意味着密钥轮换后编译缓存可复用（预期行为），但若未来编译行为意外依赖该变量，缓存可能产生陈旧产物。
  5. 测试未直接断言 cache_key_factors.json 写盘内容，仅验证 compile_factors() 返回值，端到端写盘行为依赖既有逻辑。 - 影响：对用户：使用 --api-key 或 VLLM_API_KEY 启动服务的部署不再把密钥写入容器 / 主机日志与编译缓存元数据，日志读取者无法再从启动行提取密钥；同时密钥轮换不再无效化编译缓存，可能轻微缩短轮换后的启动时间。对系统：日志与 cache_key_factors.json 的敏感信息面收敛，安全基线提升；行为变更仅限日志展示与缓存键构成，不影响参数传递、鉴权逻辑与 Rust 前端。对团队：确立「日志只留字段名、值脱敏」与「凭据类环境变量必须进 ignored_factors」两条约定，后续新增敏感参数需沿用同一模式。整体影响范围中等但杠杆高。 - 风险标记：敏感字段硬编码白名单，仅顶层字段脱敏，覆盖路径有限，缓存键语义变化

关联脉络

- 暂无明显关联 PR