PR #6542 完整报告

verl-project/verl

[rollout] fix: avoid arbitrary code execution in Qwen3 tool parser

合并时间：2026-06-07 17:49

原文链接：http://prhub.com.cn/verl-project/verl/pull/6542

执行摘要

- 一句话：修复 Qwen3 工具解析器中 eval 导致的 RCE 漏洞
- 推荐动作：值得合并。修复关键安全漏洞，代码简洁，测试完备。建议开发者关注 ast.literal_eval 的限制（不支持复杂表达式），但当前场景足够。

功能与动机

PR body 明确指出：Qwen3XMLToolParser._parse_xml_function_call 对 array 等非基本类型参数值调用 eval(param_value)，param_value 直接来自解码后的模型输出，恶意构造的工具调用可导致任意代码执行（RCE）。关联 issue #5331 跟踪同类漏洞并推荐 ast.literal_eval()。

实现拆解

1. 导入调整：在 verl/experimental/agent_loop/tool_parser.py 顶部新增 import ast。
2. 替换 eval 调用：在 convert_param_value 函数的 else 分支中，将 param_value = eval(param_value) 替换为 param_value = ast.literal_eval(param_value)，并更新日志消息中提及的转换方式。
3. 新增 CPU 测试：创建 tests/experimental/agent_loop/test_qwen3_tool_parser_on_cpu.py，包含两个测试用例：
   - test_array_literal_is_parsed：验证合法数组字面量 [1, 2, 3] 仍被正确解析。
   - test_array_param_does_not_execute_arbitrary_code：验证代码执行载荷被安全拒绝并降级为原始字符串。 测试通过模拟工具模式、直接调用 _parse_xml_function_call，无需 GPU 或 tokenizer。

关键文件：

- tests/experimental/agent_loop/test_qwen3_tool_parser_on_cpu.py（模块 工具解析；类别 test；类型 test-coverage；符号 _make_tool, _parse_single_param, test_array_literal_is_parsed, test_array_param_does_not_execute_arbitrary_code）：新增 CPU 测试，覆盖合法数组解析和 RCE 防护，确保修复正确性。
- verl/experimental/agent_loop/tool_parser.py（模块 工具解析；类别 source；类型 security-fix；符号 convert_param_value）：核心修复文件：将 eval 替换为 ast.literal_eval，修复 RCE 漏洞。

关键符号：convert_param_value

关键源码片段

tests/experimental/agent_loop/test_qwen3_tool_parser_on_cpu.py

新增 CPU 测试，覆盖合法数组解析和 RCE 防护，确保修复正确性。

```
# 测试文件：tests/experimental/agent_loop/test_qwen3_tool_parser_on_cpu.py
"""CPU tests for ``Qwen3XMLToolParser`` parameter-value conversion.

These exercise ``_parse_xml_function_call`` directly (no tokenizer / network
needed) and in particular guard against arbitrary code execution when a
non-primitive parameter (e.g. an ``array``) is parsed from untrusted model
output.
"""

import json

from verl.experimental.agent_loop.tool_parser import Qwen3XMLToolParser
from verl.tools.schemas import (
    OpenAIFunctionParametersSchema,
    OpenAIFunctionPropertySchema,
    OpenAIFunctionSchema,
    OpenAIFunctionToolSchema,
)


def _make_tool(param_name: str, param_type: str) -> OpenAIFunctionToolSchema:
    # 根据参数名和类型构建一个测试用的工具 schema
    return OpenAIFunctionToolSchema(
        type="function",
        function=OpenAIFunctionSchema(
            name="list_tool",
            description="a tool used for testing",
            parameters=OpenAIFunctionParametersSchema(
                type="object",
                properties={param_name: OpenAIFunctionPropertySchema(type=param_type)},
                required=[],
            ),
        ),
    )


def _parse_single_param(param_type: str, raw_value: str):
    # 构造一个完整的 XML 函数调用字符串，模拟模型输出中的 <function=...><parameter=...>...</
    parameter>...</function>
    parser = Qwen3XMLToolParser(tokenizer=None)
    tool = _make_tool("items", param_type)
    function_call_str = f"list_tool><parameter=items>{raw_value}</parameter>"
```

```
    result = parser._parse_xml_function_call(function_call_str, [tool])
    return result


def test_array_literal_is_parsed():
    """A legitimate array literal is still parsed into its Python value."""
    result = _parse_single_param("array", "[1, 2, 3]")
    # 合法的列表字面量应被正确解析为 JSON 数组
    assert result.arguments == '{"items": [1, 2, 3]}'


def test_array_param_does_not_execute_arbitrary_code(tmp_path):
    """An ``array`` parameter must never execute code from model output.

    Regression test: the value used to be passed to ``eval()``, allowing
    arbitrary code execution. It must now be handled by ``ast.literal_eval``,
    which rejects non-literals and degenerates to the raw string.
    """
    marker = tmp_path / "pwned.txt"
    assert not marker.exists()

    # 构造一个可能导致代码执行的 payload
    payload = f'__import__("os").system("echo pwned > {marker}")'
    result = _parse_single_param("array", payload)

    # 验证没有代码执行：标记文件未被创建
    assert not marker.exists(), "ast.literal_eval must not execute arbitrary code"
    # 验证不可解析的值被降级为原始字符串
    assert json.loads(result.arguments)["items"] == payload
```

verl/experimental/agent_loop/tool_parser.py

核心修复文件：将 eval 替换为 ast.literal_eval，修复 RCE 漏洞。

```
# 文件：verl/experimental/agent_loop/tool_parser.py
# 在文件顶部新增 import
import ast

# ... 其他代码 ...

def convert_param_value(param_value: str, param_name: str, param_config: dict, func_name:
str):
    # 根据参数类型转换值，处理基本类型（string/int/float/bool）后，对于其他类型（如 array）使用
    ast.literal_eval
    # ... 前面的类型判断代码 ...
    else:
        if param_type == "object" or param_type.startswith("dict"):
            try:
                param_value = json.loads(param_value)
                return param_value
```

```
            except Exception:
                logger.warning(
                    f"Parsed value '{param_value}' of parameter '{param_name}' is not a valid "
                    f"JSON object in tool '{func_name}', will try other methods to parse it."
                )
        try:
            # 使用 ast.literal_eval 替代 eval：参数值来自不可信的模型输出，
            # eval() 会允许任意代码执行。literal_eval 只解析 Python 字面量
            # （列表、元组、字典、数字等）。
            param_value = ast.literal_eval(param_value)
        except Exception:
            logger.warning(
                f"Parsed value '{param_value}' of parameter '{param_name}' cannot be converted "
                f"via `ast.literal_eval()` in tool '{func_name}', degenerating to string."
            )
        return param_value
```

评论区精华

无 review 评论或争议。gemini-code-assist[bot] 自动审查后表示无反馈。wuxibin89 直接批准。

- 暂无高价值评论线程

风险与影响

- 风险：变更非常集中（仅 2 个文件，源码 +6/-2），风险极低。ast.literal_eval 与 eval 的 fallback 行为一致（异常时返回原始字符串），不会破坏现有功能。测试覆盖了合法和恶意输入，回归风险小。
- 影响：影响所有使用 Qwen3XMLToolParser 的 agent 工作流，消除 RCE 漏洞，提升系统安全性。用户无感知，合法功能（如解析数组参数）保持不变。
- 风险标记：安全修复

关联脉络

- PR #5331 [security] fix: replace eval with ast.literal_eval in multiple locations: 同一漏洞类型，该 issue 推荐使用 ast.literal_eval，但未覆盖本 PR 中的 tool_parser.py 位置。