New issue

GitPilot-MCP Command Injection in run_tests Tool #38

Open

wing3e opened 3 weeks ago Owner

GitPilot-MCP Command Injection in run_tests Tool

1) CNA / Submission Type

- Submission type: Report a vulnerability (CVE ID request)
- Reporter role: Independent security researcher
- Report date: April 2, 2026

2) Reporter Contact (fill before submit)

- Reporter name: `Winegee`
- Reporter email: `Winegee@zju.edu.cn`
- Permission to share contact with vendor: `Yes`

3) Vendor / Product Identification

- Vendor: Divyanshu-hash
- Product: GitPilot-MCP
- Repository: https://github.com/Divyanshu-hash/GitPilot-MCP
- Reviewed local source path: `datasets_set/003/Divyanshu-hash-GitPilot-MCP`
- Affected component(s):
- `main.py`

4) Vulnerability Type

- CWE: CWE-78 (OS Command Injection)
- Short title: GitPilot-MCP Command Injection in run_tests Tool

5) Affected Versions

- Confirmed affected: 9ed9f153ba4158a2ad230ee4871b25130da29ffd
- Suspected affected range: versions containing the same input-to-sink flow documented below
- Fixed version: Not available at time of report (April 2, 2026)

6) Vulnerability Description

The `run_tests` tool accepts an attacker-controlled `command` argument and executes it with `shell=True`. Because the parameter is not validated or restricted to known test runners, a malicious caller can inject arbitrary shell commands.

7) Technical Root Cause

1. `run_tests(repo_path, command)` directly exposes `command` to callers.
2. `subprocess.run(command, shell=True, cwd=repo_path, ...)` executes the string through a shell.
3. No command allowlist or escaping is applied.

```
@mcp.tool()
def run_tests(repo_path: str, command: str = "pytest -q") -> dict:
    proc = subprocess.run(
        command,
        cwd=repo_path,
        shell=True,
        capture_output=True,
        text=True,
        timeout=300
    )
```

8) Attack Prerequisites

- Attacker can send requests to the vulnerable endpoint or MCP tool interface.
- Deployment does not enforce compensating controls that block the demonstrated payload.
- Vulnerable code path remains enabled in runtime configuration.

9) Proof of Concept / Reproduction Guidance

1. Invoke MCP tool `run_tests` with a malicious `command`.

2. Ensure `repo_path` points to an existing directory.
3. Verify command side effect.

PoC request:

```
{
  "jsonrpc": "2.0",
  "id": 7,
  "method": "tools/call",
  "params": {
    "name": "run_tests",
    "arguments": {
      "repo_path": "/tmp",
      "command": "pytest -q; touch /tmp/gitpilot_cmdinj_poc"
    }
  }
}
```

Verification:

```
ls -l /tmp/gitpilot_cmdinj_poc
```

10) Security Impact

- Confidentiality: High (attacker can run arbitrary shell reads).
- Integrity: High (attacker can write/modify repository and host files).
- Availability: High (attacker can disrupt CI-like execution workflows).
- Scope: Unchanged to Changed depending on surrounding deployment topology.

11) CVSS v3.1 Suggestion

- Suggested vector: `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H (9.8 for remotely exposed MCP deployments)`
- Final score should be adjusted by CNA/vendor based on real deployment exposure.

12) Workarounds / Mitigations

- Restrict access to vulnerable interfaces (network ACL, authn/authz, mTLS).
- Reject untrusted metacharacters/URLs and enforce strict server-side allowlists.
- Add regression tests for all PoC payload patterns included in this report.

13) Recommended Fix

- Remove direct execution/fetch of attacker-controlled values.
- Use safe APIs (`subprocess.run` with argument arrays and `shell=False`; strict URL policy checks for outbound requests).
- Enforce endpoint-level authentication and authorization before reaching sensitive sinks.

14) References

- Repository: https://github.com/Divyanshu-hash/GitPilot-MCP
- Reviewed source location: `datasets_set/003/Divyanshu-hash-GitPilot-MCP`
- SARIF evidence references:
- `py/mcp-command-line-injection` at `main.py:229`
- `py/mcp-path-injection` at `main.py:87`
- `py/mcp-path-injection` at `rag.py:94`
- `py/mcp-path-injection` at `rag.py:103`

15) Credits

- Discoverer: `Winegee`
- Discovery method: Static analysis (CodeQL/SARIF) plus source-code audit

16) Additional Notes for Form Mapping

- Issue status at report time: source-code confirmed in local dataset and exploitation path documented with explicit PoC.
- Dynamic verification was represented through deterministic command/URL payloads suitable for lab reproduction.
- Version-range precision should be finalized by maintainer release history before disclosure.

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants