Path Traversal (Zip Slip) in OpenMRS Module Upload

High **ibacher** published **GHSA-78fc-9688-w8xw** 2 days ago

Package

org.openmrs:openmrs-web (Maven)

Affected versions

<= 2.7.8, >= 2.8.0 <= 2.8.5

Patched versions

> 2.7.8 < 2.8.0, > 2.8.5

Description

Affected Versions

version ≤ 2.7.8 (latest version at time of disclosure)

https://github.com/openmrs/openmrs-core

Impact

The endpoint `POST /openmrs/ws/rest/v1/module` is vulnerable to a path traversal (Zip Slip) attack. An authenticated attacker can upload a crafted `.omod` archive containing ZIP entries with directory traversal sequences. Upon automatic extraction by the server, the incomplete path validation in `WebModuleUtil.startModule()` fails to prevent entries such as `web/module/../../../../malicious.jsp` from being written outside the intended module directory. If the traversal target falls within the web application root (e.g., `/usr/local/tomcat/webapps/openmrs/`), the attacker achieves arbitrary file write and subsequent Remote Code Execution.

Notably, other extraction methods in the same codebase (`ModuleUtil.expandJar()`, `TestInstallUtil.addZippedTestModules()`) are properly protected with `normalize().startsWith()` checks — this vulnerability is an oversight where the same fix was not applied.

Furthermore, the `module.allow_web_admin` runtime property, which is intended to restrict administrators from managing modules via the web interface, only gates the Legacy UI controller entry point. The REST API endpoint `POST /openmrs/ws/rest/v1/module` does not check this property, allowing this restriction to be fully bypassed.

Steps to Reproduce

1. Construct a malicious `.omod` file (which is a ZIP/JAR archive) containing a ZIP entry with a path traversal payload in its entry name, such as `web/module/../../../../<target_filename>`. Upload this file to `POST /openmrs/ws/rest/v1/module` with valid admin credentials via Basic Auth.

2. The server parses and loads the module. During `WebModuleUtil.startModule()`, entries under `web/module/` are automatically extracted. The existing check

`Paths.get(name).startsWith("..")` only blocks entries beginning with `..`, so an entry starting with `web/module/` passes the check. The `../` sequences in the remaining path cause the file to be written outside the intended `WEB-INF/view/module/` directory — for example, into the web application root at `/usr/local/tomcat/webapps/openmrs/`.

```
event-4.0.0.omod                    ordertemplates-2.0.0.omod   webservices.rest-3.1.0.omod
sh-4.2$ ls /usr/local/tomcat/webapps/openmrs/
META-INF  error.html        images     initialsetup     openmrs.css  openmrs_green.css   openmrs_orange.css  shell.jsp
WEB-INF   errorhandler.jsp  index.htm  memoryUsage.jsp  openmrs.js   openmrs_legacy.css  openmrs_purple.css  style.css
sh-4.2$
```

3. The traversed file is now accessible under the web application root. If the written file is a JSP script, accessing it via the browser triggers server-side execution, achieving RCE.

Root Cause Analysis

The vulnerability exists in `WebModuleUtil.startModule()` (`web/src/main/java/org/openmrs/module/web/WebModuleUtil.java`).

Vulnerable code:

```
Enumeration<JarEntry> entries = jarFile.entries();
while (entries.hasMoreElements()) {
    JarEntry entry = entries.nextElement();
    String name = entry.getName();

    // ❌ Incomplete check — only blocks entries starting with ".."
    if (Paths.get(name).startsWith("..")) {
        throw new UnsupportedOperationException("...");
    }

    if (name.startsWith("web/module/")) {
        String filepath = name.substring(11);
        StringBuilder absPath = new StringBuilder(realPath + "/WEB-INF");
        absPath.append("/view/module/");
        absPath.append(mod.getModuleIdAsPath()).append("/").append(filepath);

        // ❌ No normalize() or startsWith() boundary check before writing
        File outFile = new File(absPath.toString().replace("/", File.separator));
        outStream = new FileOutputStream(outFile, false);
        inStream = jarFile.getInputStream(entry);
        OpenmrsUtil.copyFile(inStream, outStream);
    }
}
```

Why the check fails: For an entry named `web/module/foo/../../../../evil.jsp`, `Paths.get(name)` starts with `web`, not `..`, so the check passes. After `name.substring(11)`, the filepath `foo/../../../../evil.jsp` is concatenated directly into the output path without normalization, resulting in a write outside the intended directory.

Correctly protected code in the same codebase:

`ModuleUtil.expandJar()`:

```
// ✅ Correct — uses normalize().startsWith()
if (!parent.toPath().normalize().startsWith(docBase)) {
    throw new UnsupportedOperationException("...");
}
```

`TestInstallUtil.addZippedTestModules()`:

```
// ✅ Correct — uses normalize().startsWith()
if (!zipEntryFile.toPath().normalize().startsWith(moduleRepository.toPath().normalize()))
    throw new IOException("Bad zip entry");
}
```

The fix pattern is already known and applied elsewhere in the codebase. `WebModuleUtil.startModule()` is an oversight.

Bypass of `module.allow_web_admin`

The `module.allow_web_admin` property only restricts module operations at the Legacy UI layer (`ModuleListController`). The REST API endpoint does not consult this property:

```
Legacy UI:  POST /admin/modules/moduleList.form → allowAdmin() check → [BLOCKED]
REST API:   POST /ws/rest/v1/module             → No allowAdmin() check → [ALLOWED]
                ↓
        ModuleFactory.loadModule()
                ↓
        WebModuleUtil.startModule()   ← Zip Slip here, no allowAdmin check
                ↓
        FileOutputStream.write()      ← Arbitrary file write
```

Remediation

Add `normalize().startsWith()` boundary validation before writing, consistent with the existing pattern in `ModuleUtil.expandJar()`:

```
File outFile = new File(absPath.toString().replace("/", File.separator));

// ✅ Add this check
if (!outFile.toPath().normalize().startsWith(
        Paths.get(realPath, "WEB-INF").normalize())) {
    throw new UnsupportedOperationException(
        "Zip entry '" + name + "' would be written outside the allowed directory.");
}
```

Additionally, enforce the `module.allow_web_admin` restriction consistently across all module upload entry points, including the REST API.

Severity

High 8.7 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	High
User interaction	None
Scope	Changed
Confidentiality	High
Integrity	High
Availability	None

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:C/C:H/I:H/A:N

CVE ID

CVE-2026-40076

Weaknesses

No CWEs

Credits

Arron-bit Reporter