Often Misused: File Upload Mitigation SOP

# Development Mitigation SOP

Regardless of the language a program is written in, the most devastating attacks often involve remote code execution, whereby an attacker succeeds in executing malicious code in the program’s context. If attackers are allowed to upload files to a directory that is accessible from the Web and cause these files to be passed to a code interpreter (e.g. JSP/APSX/PHP), then they can cause malicious code contained in these files to execute on the server.

# Defense Against [DEFECT]

Do not accept attachments if they can be avoided. If a program must accept attachments, then restrict the ability of an attacker to supply malicious content by only accepting the specific types of content the program expects. Most attacks that rely on uploaded content require that attackers be able to supply content of their choosing. Placing restrictions on the content the program will accept will greatly limit the range of possible attacks. Check file names, extensions, and file content to make sure they are all expected and acceptable for use by the application. Make it difficult for the attacker to determine the name and location of uploaded files. Such solutions are often program-specific and vary from storing uploaded files in a directory with a name generated from a strong random value when the program is initialized to assigning each upload file a random name and tracking them with entries in a database.

# Examples

…

Optional<Object> readFile(MultipartFile multipartFile) {

List<String> bucketCategoryList = new ArrayList<>();

bucketCategoryList.addAll(worktypeCapacityCategoryService.getWorktypeCategoryMapping().keySet());

…

## Explanation

A parameter of type org.springframework.web.multipart.MultipartFile in the excerpt is used by the Spring MVC framework to set uploaded files. Permitting users to upload files can allow attackers to inject dangerous content or malicious code to run on the server.

## Example

…

public String moveDocument(@RequestParm MultipartFile csvFile, @RequestParm String pageId, @RequestParm String fromFukeNumber, @RequestParm String toFileNumber, @RequestParm String commaSeparatedDocumentIds, HttpSession httpSession, Model model) throws IOException {

…

## Recommendation

The code below takes in a CSV that is validated in multiple places. The number of headers is validated in multiple places. The values are scrubbed for whitespace characters by constructors to protect against log forging.

**Resources**

1. [Fortify Taxonomy: Software Security Errors](https://vulncat.fortify.com/en/weakness?category=Often+Misused%3BPrivacy+Violation&pci1.2=Requirement+8.4%3BRequirement+6.3.1.5&pci3.0=Requirement+6.5.1&pci3.1=Requirement+2.2.4&wasc2_00=Insufficient+Authentication+(WASC-01)&stig39=APP3120+CAT+II&stig43=APSC-DV-002570+CAT+II%3BAPSC-DV-000640+CAT+II&sans2010=Risky+Resource+Management+-+CWE+ID+805)