# CWE Detail – CWE-434

## Description

The product allows the upload or transfer of dangerous file types that are automatically processed within its environment.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2023-5227: PHP-based FAQ management app does not check the MIME type for uploaded images

**•** CVE-2001-0901: Web-based mail product stores ".shtml" attachments that could contain SSI

**•** CVE-2002-1841: PHP upload does not restrict file types

**•** CVE-2005-1868: upload and execution of .php file

**•** CVE-2005-1881: upload file with dangerous extension

**•** CVE-2005-0254: program does not restrict file types

**•** CVE-2004-2262: improper type checking of uploaded files

**•** CVE-2006-4558: Double "php" extension leaves an active php extension in the generated filename.

**•** CVE-2006-6994: ASP program allows upload of .asp files by bypassing client-side checks

**•** CVE-2005-3288: ASP file upload

**•** CVE-2006-2428: ASP file upload

## Related Attack Patterns (CAPEC)

* CAPEC-1

## Attack TTPs

**•** T1574.010: Services File Permissions Weakness (Tactics: persistence, privilege-escalation, defense-evasion)

## Modes of Introduction

**•** Implementation: N/A

**•** Architecture and Design: OMISSION: This weakness is caused by missing a security tactic during the architecture and design phase.

## Common Consequences

**•** Impact: Execute Unauthorized Code or Commands — Notes: Arbitrary code execution is possible if an uploaded file is interpreted and executed as code by the recipient. This is especially true for web-server extensions such as .asp and .php because these file types are often treated as automatically executable, even when file system permissions do not specify execution. For example, in Unix environments, programs typically cannot run unless the execute bit is set, but PHP programs may be executed by the web server without directly invoking them on the operating system.

## Potential Mitigations

**•** Architecture and Design: Generate a new, unique filename for an uploaded file instead of using the user-supplied filename, so that no external input is used at all.[REF-422] [REF-423] (Effectiveness: N/A)

**•** Architecture and Design: When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. (Effectiveness: N/A)

**•** Architecture and Design: Consider storing the uploaded files outside of the web document root entirely. Then, use other mechanisms to deliver the files dynamically. [REF-423] (Effectiveness: N/A)

**•** Implementation: Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. For example, limiting filenames to alphanumeric characters can help to restrict the introduction of unintended file extensions. (Effectiveness: N/A)

**•** Architecture and Design: Define a very limited set of allowable extensions and only generate filenames that end in these extensions. Consider the possibility of XSS (CWE-79) before allowing .html or .htm file types. (Effectiveness: N/A)

**•** Implementation: Ensure that only one extension is used in the filename. Some web servers, including some versions of Apache, may process files based on inner extensions so that "filename.php.gif" is fed to the PHP interpreter.[REF-422] [REF-423] (Effectiveness: N/A)

**•** Implementation: When running on a web server that supports case-insensitive filenames, perform case-insensitive evaluations of the extensions that are provided. (Effectiveness: N/A)

**•** Architecture and Design: For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. (Effectiveness: N/A)

**•** Implementation: Do not rely exclusively on sanity checks of file contents to ensure that the file is of the expected type and size. It may be possible for an attacker to hide code in some file segments that will still be executed by the server. For example, GIF images may contain a free-form comments field. (Effectiveness: N/A)

**•** Implementation: Do not rely exclusively on the MIME content type or filename attribute when determining how to render a file. Validating the MIME content type and ensuring that it matches the extension is only a partial solution. (Effectiveness: N/A)

**•** Architecture and Design: Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations. (Effectiveness: N/A)

**•** Architecture and Design: Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software. OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations. This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise. Be careful to avoid CWE-243 and other weaknesses related to jails. (Effectiveness: Limited)

## Applicable Platforms

**•** ASP.NET (Class: None, Prevalence: Sometimes)

**•** PHP (Class: None, Prevalence: Often)

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** Once submitted, the form above sends the file to upload\_picture.php on the web server. PHP stores the file in a temporary location until it is retrieved (or discarded) by the server side code. In this example, the file is moved to a more permanent pictures/ directory.

**•** When submitted the Java servlet's doPost method will receive the request, extract the name of the file from the Http request header, read the file contents from the request and output the file to the local upload directory.

## Notes

**•** Relationship: This can have a chaining relationship with incomplete denylist / permissive allowlist errors when the product tries, but fails, to properly limit which types of files are allowed (CWE-183, CWE-184). This can also overlap multiple interpretation errors for intermediaries, e.g. anti-virus products that do not remove or quarantine attachments with certain file extensions that can be processed by client systems.