

# Flawed Identification of File Type

---

## Introduction

This scenario demonstrates how unreliable **client-side-only file type identification** can lead to a **file type validation vulnerability**. Modern websites rarely allow unrestricted file uploads, but not every developer realizes that **relying on client-side checks or trusting values sent in an HTTP POST request** is a serious risk.

Most HTML forms use the `POST` method with the `application/x-www-form-urlencoded` content type.

More advanced forms can accept file uploads using `multipart/form-data`. In this case, the file type is determined on the client side, and the tagged file is sent via `POST` to the server. If the server validates only the header value and not the actual file content, this creates a security gap.

If this mechanism is not properly secured, it can be exploited. Such a vulnerability allows an attacker to upload unauthorized files, bypass intended restrictions, or modify server behavior.

## Examples

POST method for form containing fields for uploading an **image**, providing a **description** of it, and entering **username**.

```
POST /images HTTP/1.1
Host: normal-website.com
Content-Length: 12345
Content-Type: multipart/form-data; boundary=-----
---012345678901234567890123456

-----012345678901234567890123456
Content-Disposition: form-data; name="image"; filename="example.jpg"
Content-Type: image/jpeg

[...binary content of example.jpg...]

-----012345678901234567890123456
Content-Disposition: form-data; name="description"

This is an interesting description of my image.

-----012345678901234567890123456
Content-Disposition: form-data; name="username"

wiener
-----012345678901234567890123456--
```

### What a thread actor can do?

Thread actor can attempt to modify the `Content-Type` value of the file upload field. If the server implicitly trusts this value without performing its own validation, we can upload arbitrary files despite the intended restrictions.

## **LAB**

**goal:** examine a vulnerable file upload mechanism that incorrectly trusts file type validation based solely on the POST request.

**requirements:** Internet connection, BurpSuit, PortSwigger account

**LAB:** [link](#)

↓ hints below ↓

## Hints

**hint 1:** find unrestricted file upload function and try to exploit it.

**hint 2:** based on last lab, prepare payload file and upload it.

**hint 3:** intercept file upload POST and modify it's Content-Type field to be allowed by server (or use Repeater ).

**hint 4:** using Proxy | HTTP history find response to Your GET request generated by uploading malicious file.