by Azmi Alsarayrah

PHP Web Application Handling File Uploads

```
<?php
if ($_FILES['file']['error'] == UPLOAD_ERR_OK) {
    $upload_dir = 'uploads/';
    $upload_file = $upload_dir . basename($_FILES['file']['name']);
    if (move_uploaded_file($_FILES['file']['tmp_name'], $upload_file)) {
        echo "File is valid, and was successfully uploaded.\n";
    } else {
        echo "Possible file upload attack!\n";
    }
} else {
    echo "File upload error.\n";
}
?>
```

vulnerabilities

1. **Lack of Input Validation**: The code doesn't validate the uploaded file type, size, or extension. This can lead to potential security issues, such as:
    - File type injection: An attacker could upload a malicious file with a different extension, potentially executing it as a script.
2. **Lack of Sanitization**: The code doesn't sanitize the uploaded file name, which could lead to:
    - Directory traversal attacks: An attacker could upload a file with a malicious path, potentially accessing sensitive files or directories.
3. **Lack of Secure File Storage**: The code stores files in a directory with a predictable name (`uploads/`). This could lead to:
    - File exposure: An attacker could guess the directory name and access sensitive files.
    - Directory traversal attacks: An attacker could access files outside the `uploads/` directory.

Recommendations

1. Validate file type:

```
$allowed_types = ['image/jpeg', 'image/png', 'application/pdf'];
if (!in_array($_FILES['file']['type'], $allowed_types)) {
    die("Invalid file type.");
}
```

to only allow specific file types

2. use generated file name:

```
$upload_dir = 'uploads/';
$upload_file = $upload_dir . uniqid() . '_' . basename($_FILES['file']
['name']);
```

3. limit file size:

```
if ($_FILES['file']['size'] > 2000000) {
    die("File is too large.");
}
```

for this example to only allow 2MB