Einheit 15 – 06.12.2017 (Woche 49)

bereitzustellendes Material	• Unterrichts-Unterlagen • Code-Beispiele, Demo-Programm (git-branch „dateizugriff“)
Ziele	• Sie wissen, wie Sie in PHP Dateien hochladen und speichern • Sie wissen, wie Sie in PHP Dateien ausliefern • Sie kennen die Problematik von Direkt-Links via Webserver auf schützenswerte Dateien • Sie kennen Schutzmechanismen, um das direkte Ausliefern von Dateien zu verhindern
Hausaufgaben	•

Startzeit	**Dauer**	**Thema, Details**
08:20	45min	Unterricht zum Thema
09:05	**5min**	**Pause**
09:10	rest	Selbständige Arbeit am Projekt
09:55	**Ende**	

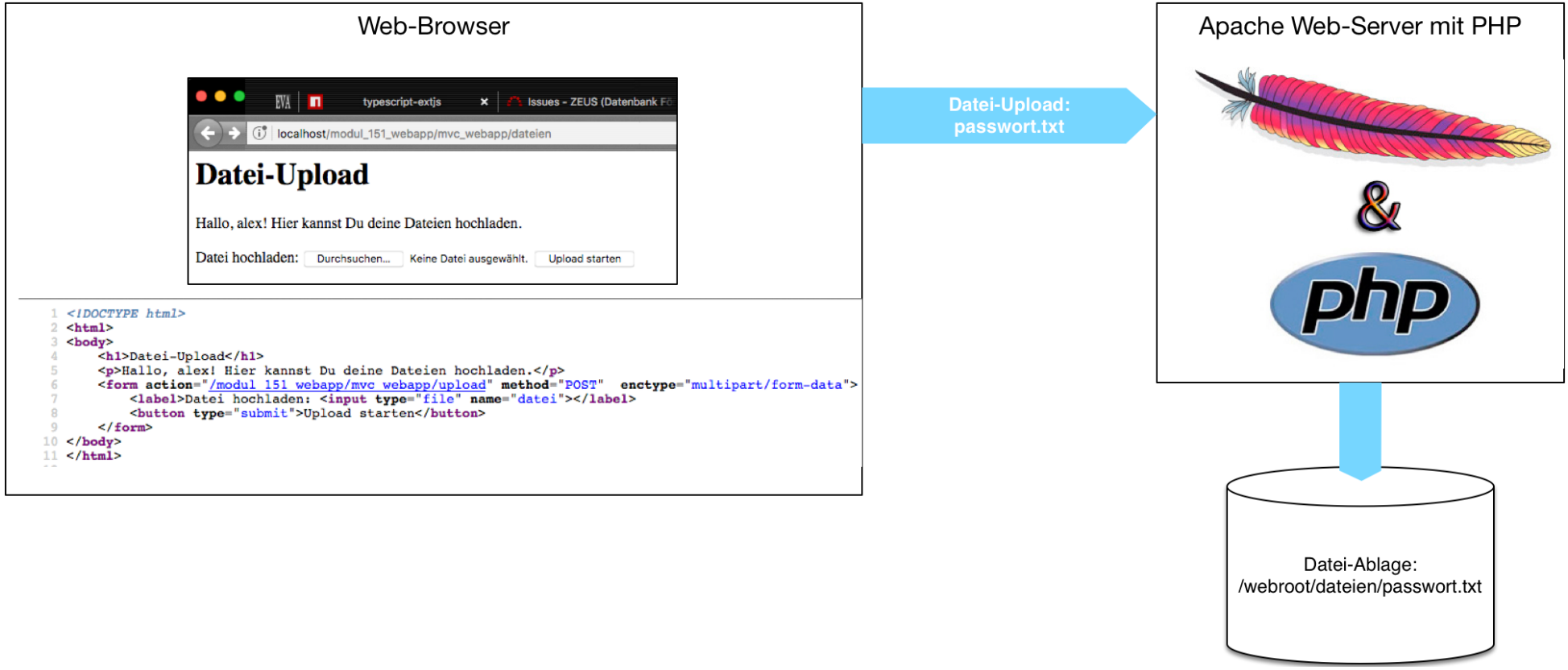
Material:

- Unterrichts-Unterlagen
- vorbereitete Code-Beispiele (git-branch „dateizugriff“)

Wir laden Dateien in unsere Applikation hoch

Um Dateien in einer Web-Applikation hochzuladen, bedarf es:

- im Frontend (HTML): Ein HTML-Formular mit Encoding **„multipart/form-data“** und mind. einem **Dateifeld (<input type=“file“ />)**
- im Backend (PHP): ein Programmteil, welcher die Datei entgegennimmt und speichert:



Das zugehörige HTML-Formular sieht so aus:

```
<!DOCTYPE html>
<html>
<body>
    <h1>Datei-Upload</h1>
    <p>Hallo, alex! Hier kannst Du deine Dateien hochladen.</p>
    <form action="/modul_151_webapp/mvc_webapp/upload" method="POST" enctype="multipart/form-data">
        <label>Datei hochladen: <input type="file" name="datei" /></label>
        <button type="submit">Upload starten</button>
    </form>
</body>
</html>
```

Wichtig sind hier die Angaben:

- action=““: Wohin wird der Upload-Request / die Datei geschickt?
- method=“POST“: Dateien können nur via http POST hochgeladen werden
- enctype=“multipart/form-data“: Dies definiert das Format des POST-Requests und ist für File-Uploads zwingend.

Nach Absenden (Button „Upload starten“) nimmt unser PHP-Script die Datei entgegen und speichert sie ab:

```
public function upload(Request $req) {
    header('Content-type: text/plain');
    // Datei-Uploads in PHP werden im Super-Global "$_FILES" geliefert:
    // Jedes File-Form-Field hat einen Dictionnary-Eintrag, in diesem Fall $_FILES['datei']:
    var_dump($_FILES['datei']);

    // Prüfen, ob das File korrekt hochgeladen wurde:
    $ok = $_FILES['datei']['error'] === 0;
    if ($ok) {
        // Datei sicher speichern:
        // tmp_name beinhaltet den Datei-Pfad zum hochgeladenen File:
        $upload_pfad = $_FILES['datei']['tmp_name'];
        // 'name' beinhaltet den Originalnamen des Files
        $upload_filename = $_FILES['datei']['name'];

        // BEISPIEL: Zielpfad zusammenstellen / ermitteln, hier wird das File in ein Unterverzeichnis
        // mit dem Usernamen gespeichert:
        $username = $_SESSION['userinfo']['login'];
        $datei_dir = __DIR__.'/../../../uploads/'.$username;
        $zielpfad = $datei_dir.'/'.$upload_filename;
        if (!is_dir($datei_dir)) {
            mkdir($datei_dir);
        }
        // move_uploaded_file prüft, ob das angegebene File ein korrektes Upload-File ist
        // und verschiebt es an den angegebenen Zielpfad:
        move_uploaded_file($upload_pfad,$zielpfad);
        echo "Datei gespeichert in {$zielpfad}";
    }
}
```

- Dateien von Formularen werden in PHP im Super-Globals-Array „**$_FILES**“ geliefert.
- Diese Superglobal stellt Informationen zu den hochgeladenen Dateien bereit (Upload-Pfad, originaler Dateiname, ev. Fehler). Siehe http://php.net/manual/de/features.file-upload.post-method.php.
- Ein hochgeladenes File kann dann mit der PHP-Funktion „**move_uploaded_file()**“ in das gewünschte Zielverzeichnis / Zielname verschoben werden. Siehe http://php.net/manual/de/function.move-uploaded-file.php.

→ Da das File nun im Webroot liegt, kann es auch direkt vom Web-Server, ohne den „Umweg“ über PHP, ausgeliefert werden. Jeder, der den Web-Pfad kennt, kann das File downloaden. Dies ist unter Umständen problematisch, wenn die Dateien sensitive Informationen beinhalten.

Speichern von Verweisen der Dateien in einer Datenbank-Tabelle

Je nach Problemstellung macht es Sinn, Meta-Informationen zu den gespeicherten Dateien in einer Datenbank-Tabelle festzuhalten, z.B.:

- um den lokalen Pfad zu speichern / den ursprünglichen Dateinamen festzuhalten
- um Dateigrössen zu berechnen (bsp: „Quota“ pro User)
- um Autorisation zu ermöglichen: wem gehört die Datei, wer hat Zugriff darauf
- um Informationen über die Datei zu speichern (Bilddimension, MIME-Typ, ….)
- um Zeitstempel (z.B. Hochladedatum, für Validitätsprüfung) festzuhalten

Nach dem Hochladen verzeichnen Sie die Datei in der Datenbank, z.B.:

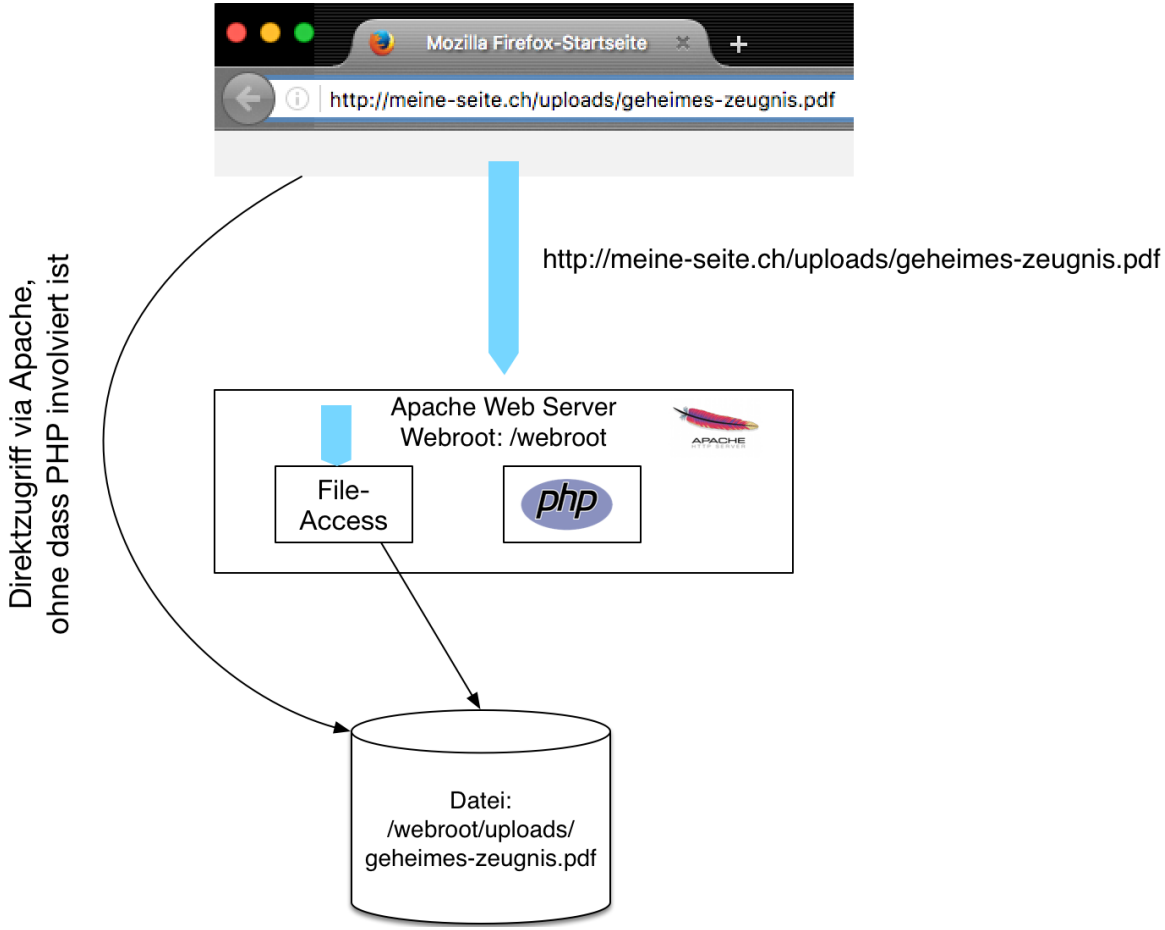
```
// Orig-Filename: $_FILES['input-name']['name']
// Dateigrösse  : $_FILES['input-name']['size']

// SQL für Beispieltabelle „files“, siehe unten:
INSERT INTO files (`orig_filename`,`local_path`,`size`,`benutzer_id`,`uploaded_at`)
VALUES ('IMG_3436782.png','3/88.png',33898476,3,NOW());
```

id	orig_filename	local_path	size	benutzer_id	uploaded_at
1	My-uploaded-File.pdf	3/1.pdf	34871	3	2017-12-13 06:26:48
2	IMG_55987644.PNG	3/2.pdf	1784632	3	2017-12-13 06:27:47

Problematik: Direktzugriffe auf Dateien

Oben haben wir gesehen, dass die Dateien beim Hochladen in einem Folder innerhalb des Webroot des Webservers abgelegt wurden. Dateien im Webroot des Webservers können ohne weitere Massnahme Seitens des Entwicklers auch wieder direkt vom Webserver via URL ausgeliefert werden:



Dies ist problematisch:

- Jeder, der den Link kennt, kann die Dateien herunterladen
- Die Dateien können sogar von Suchrobotern indiziert und gelesen werden
- Sensitive Daten sind so völlig ungeschützt.

Für sensitive oder nicht öffentliche Daten muss dies verhindert werden. Es muss also die **Autorisation** sichergestellt werden.
Dazu sind folgende Massnahmen zu treffen:

1. Das Ausliefern der Dateien direkt via Web-Server muss verhindert werden. Dies kann auf folgende Arten geschehen:
   - durch Zugriffsberechtigung durch den Webserver (Stichwort: „Deny from all“ in Apache-.htaccess-Direktive)
     Beispiel in Apache **.htaccess**-file:
     ```
     Require all denied
     ```
   - durch Platzieren der Dateien **ausserhalb des Webserver-Webroots**
2. Die Auslieferung muss nun durch die Applikation (in diesem Fall durch PHP) erfolgen: Somit hat man die Möglichkeit, Zugriffsberechtigungen zu prüfen.

Wir liefern Dateien via PHP aus

Die Auslieferung durch die server-seitige Applikation, in diesem Beispiel von PHP, hat folgende Vorteile:

- Die **Authentizität** des Zugriffs kann geprüft werden (Ist der Benutzer angemeldet?)
- Die **Autorisation** kann sichergestellt werden (Darf der angemeldete Benutzer die Datei lesen?)

Die Auslieferung erfolgt somit nicht mehr direkt durch den Web-Server, sondern durch unsere Applikation. Da wir nun nicht mehr direkt die Datei in der URL aufrufen, sondern unser serverseitiges Script / URL, muss die angeforderte Datei anderweitig angegeben werden. Im Beispiel wird dazu ein HTTP-GET-Parameter verwendet:

```
# Request: http://meine-seite.ch/download.php?filename=geheimes-zeugnis.pdf
<?php
    // download.php (oder ähnlich, je nach URL / Backend-Framework):
    // Dateiname aus Request lesen und korrekten Pfad ermitteln:
    $filename = $_GET['filename'];
    $pfad = '/pfad/zu/den/dateien/'.$filename;

    // Benutzer-Authentizität und -Autorisation prüfen:
    if (user_is_logged_in_and_can_access_the_file($filename)) {
        // Content-Type ermitteln:
        $type = mime_content_type($pfad);

        // Datei mit korrekten Headern ausliefern:
        header("Content-Type: $type");
        header("Content-Disposition: attachment; filename=\"$pfad\"");
        // Ausgeben der Datei:
        readfile($pfad);
    } else {
        echo "Zugriff verweigert!";
    }
```

Das sehr minimalistische Beispiel oben zeigt den Vorgang für eine Auslieferung via PHP:

1. Der Benutzer ruft die URL zum Download-Script auf, und übermittelt den gewünschten Dateinamen als GET-Parameter
2. Der Dateiname wird aus den Request-Parametern ermittelt und der tatsächliche Dateipfad im Dateisystem ermittelt.
3. Die Zugriffs-Authentizität und Autorisation wird geprüft (bsp: Ist der User eingeloggt, und darf dieser die Datei downloaden?)
4. Die Datei wird ausgeliefert (php: readfile). Damit der Browser diesen Download korrekt entgegennehmen kann, muss der Datei-Typ („Mime-Type“, Internet Media Type, siehe https://de.wikipedia.org/wiki/Internet_Media_Type) und die Download-Anweisung (Content-Disposition-Header) als HTTP-Response-Header ausgegeben werden.

FRAGE: Was beherbergt dieser oben gezeigte Mechanismus für ein Problem / für Gefahren?

Lösung: Ohne weitere Prüfung / Normalisierung des eingelesenen Filenamens vom Frontend ist hier eine **Directory Traversal-Attacke** möglich.

Wir behandeln Angriffsmöglichkeiten auf Web-Applikationen im nächsten Kapitel.