# Laborator 07 - Securitate II

Una din componentele cele mai sensibile ale unei aplicații web este **baza de date**. Aceasta reține informație obținută cumulativ, în timp, care valorifică (financiar, informațional) aplicația în sine. Exemple de astfel de informații:

- utilizatori autentificați;
- posturi (în cazul forumurilor);
- produse și comentarii la produse;

Pierderea acestor informații produce, pe lângă nefuncționare, devalorizarea aplicației în sine. Spre exemplu, valoarea unei aplicați de *social-networking* este dată (pe lângă funcționalități, implementare, design, etc), de numărul de persoane care o folosesc, mai general de cantitatea de informații pe care aplicația o pune la dispoziție. Or, această informație este de valoare **critică** pentru o aplicație, și ea trebuie protejată.

Un prim pas este realizarea de *back-up*-uri regulate ale informațiilor. În continuare, vom examina alte situații vulnerabile care conduc la compromiterea bazei de date, și solutii de evitare a acestora.

Soluții de protejare:

- limitarea accesului la anumite fișiere de pe server; în Apache, acest lucru se poate face editând httpd.conf astfel:

```
<Files ~ "\.inc$">
    Order allow,deny
    Deny from all
</Files>
```

- folosirea unei extensii care *forțează evaluarea script*-ului, spre deosebire de afișarea lui ca atare

## Expunerea de informații în fișiere

Majoritatea aplicațiilor web care interacționează cu baze de date rețin informațiile de conectare în fișiere auxiliare. Tema 1 folosește un astfel de fișier. Exemplu:

```
define ('ADDRESS', 'localhost');
define ('DATABASE', 'tema1');
define ('USERNAME' , 'root');
define ('PASSWORD', '');
```

Să presupunem că fișierul care conține codul de mai sus se numește **`constants.inc`**, și că se află în directorul **`root`** al aplicației. Cel mai probabil, acesta va fi inclus de un script php, astfel: `require_once('constants.inc');`. Dacă însă un utilizator intuiește (sau alege la întâmplare) numele acestui fișier, îl poate accesa astfel:

```
http://localhost/path/to/constants.inc
```

Cum nici browserul nici serverul nu au informații despre extensia *.inc, conținutul acestui fișier este întors utilizatorului ca și text. Consecințele sunt intuitive. Username-ul și parola pentru conectarea la baza de date sunt compromise, și implicit conținutul acesteia.

## SQL Injection

SQL Injection se referă la *injectarea* de cod SQL malițios la rularea unui query.
Fie următorul script care construiește un query pe baza unui formular primit:

```
$query = "INSERT
          INTO   users (username,
                        password,
                        email)
          VALUES ('{$_POST['username']}',
                  '$autoPassword',
                  '{$_POST['email']}')";
```

Variabila `$autoPassword` reprezintă o parolă generată automat, care poate fi trimisă utilizatorului via email, la crearea contului. Să afișăm acest query pentru niște valori convenționale trimise prin formular:

```
INSERT INTO users (username, password, email) VALUES ('user', '6d0f846348a856321729', 'email')
```

Ce se întâmplă însă dacă un utilizator completează câmpul `username` cu următoarea valoare:

```
bad_guy', 'mypass', ''), ('good_guy
```



- Repartizare teme
- Catalog PW

**Laboratoare**

- Laborator 01 - Introducere
- Laborator 02 - BD in PHP
- Laborator 03 - Arrays, Magic Methods
- Laborator 04 - (X)HTML, CSS
- Laborator 05 - Formulare HTML, Persistența Datelor
- Laborator 06 - Securitate I
- Laborator 07 - Securitate II
- Laborator 08 - JavaScript
- Laborator 09 - DOM scripting
- Laborator 10 - AJAX
- Laborator 11 - Web Frameworks

**Teme**

- Tema 01 - API pentru BD
- Tema 02 - Template System & Controller
- Tema 03 - Generator de formulare
- Tema 04 - Tree Web UI

Login

Dacă afișăm query-ul executat de script, obținem:

```
INSERT INTO users (username, password, email) VALUES ('bad_guy', 'mypass', ''), ('good_guy', '6d0f8
```

Efectul este vizibil: un utilizator nou este adăugat cu o parolă deja setată.

Aparent, rezultatul acestei vulnerabilități nu este foarte grav, însă putem considera următorul context:

- în locul funcției **query** (folosită uzual în laborator), folosim funcția **multi_query**, care permite rularea de instrucțiuni multiple
- trimitem în locul unui nume de utilizator șirul:

```
some_guy', '', '');  DROP TABLE criticaltable; --
```

Query-ul produs va fi următorul:

```
INSERT INTO users (username, password, email) VALUES ('some_guy', '', ''); DROP TABLE criticaltable
```

**Observații:**

- Caracterele – reprezintă comentarii; comentariile anulează restul expresiei SQL
- În ipoteza existenței tabelului `criticalTable`, acesta va fi șters
- Mai grav, instrucțiunea SQL *DROP TABLE …* putea fi înlocuită cu *DROP DATABASE …*, având un efect ușor de imaginat.
- Chiar și în cazul în care folosim funcția **query**, care execută un singur query, SQL Injection poate fi folosit în formule (de sintaxă) mai complicate, pentru a produce efecte grave asupra bazei de date (spre exemplu: extragerea de informații)

**Modalități de protejare:**

- filtrarea **oricărei** informații provenite din formulare, indiferent de metoda folosită (acceptarea valorilor care respectă un pattern prestabilit, și respingerea a orice nu respectă acest pattern)
- folosirea funcției membru **real_escape_string** din clasa **mysqli**; detalii: http://www.php.net/manual/en/mysqli.real-escape-string.php

## Session Hijacking

Un astfel de atac este posibil, dacă **session id**-ul unui utilizator este compromis. Acest lucru este foarte posibil, ținând cont de faptul ca **session-id**-ul este direct vizibil. Iată un exemplu de header HTTP, care expune **session_id**-ul. Header-ul provine de la un script care setează o sesiune:

```
HTTP/1.1 200 OK
Date: Wed, 17 Mar 2010 18:31:47 GMT
Server: Apache/2.2.11 (Win32) PHP/5.2.8 X-Powered-By: PHP/5.2.8
Set-Cookie: PHPSESSID=ls2pd33foiidr5ld6gkpqo94d0; path=/ Expires: Thu, 19 Nov 1981 08:52:00 GMT
Content-Length: 0
Connection: close
Content-Type: text/html
```

Observați linia `PHPSESSID=ls2pd33foiidr5ld6gkpqo94d0;`, care expune **session id**-ul. Incluzând acest **session id** într-o cerere HTTP, un atacator poate obține credențialele utilizatorului pe aplicația web vulnerabilă.

## Session Fixation

Ca și **Session Hijacking**, **Session Fixation** se bazează pe folosirea **session id**-ului unui utilizator, însă fără a-l fura, ci **impunându-l**. Un utilizator poate accesa un link care îi **fixează** un anumit **session id**, care apoi e folosit ca id de sesiune, la autentificarea pe o aplicație. Folosind id-ul **fixat**, atacatorul poate beneficia pe aplicația vulnerabilă, de privilegiile utilizatorului.
Detalii despre **Session Fixation**: http://en.wikipedia.org/wiki/Session_fixation

## Exercitii

- **(1.0p)** Creați un fișier **constants.inc**, într-un director oarecare, și încercați să-l accesați. Experimentați diverse soluții pentru protejarea conținutului fișierului.
- **SQL Injection**:
  - **(2.0p)** Testați cele două exemple descrise în capitolul SQL Injection. Folosiți **real_escape_string** pentru protejarea script-ului.
  - **(2.0p)** Scrieți un mecanism general de filtrare a datelor. Introduceți tipuri de date acceptate: `string`, `number`, `email`. Verificați conținutul unui formular trimis, verificând *tipul* valorii trimise.
- **Session Hijacking**:
  - **(1.0)p** Scrieți un script **session.php** care setează o sesiune corespunzătoare pentru un utilizator autentificat (omiteți verificările de username și parolă, pentru a putea simula "furtul" id-ului de sesiune)
  - Scrieți un script **secret.php** care afișează un mesaj "secret" utilizatorului autentificat, și un mesaj "acces interzis", pentru utilizatorul ne-logat (verificarea se face pe baza sesiunii)
  - **(3.0p)** Scrieți un script care realizează o cerere HTTP via portul 80 către **session.php**, extrage **session-id**-ul primit, și folosindu-l, realizează o altă cerere către **secret.php**. Această cerere va include **session_id**-ul, și pe baza ei puteți obține acces la mesajul secret.

- **(3.0p)** Propuneți metode de prevenire pentru **Session Hijacking**;
- Pentru rezolvarea ultimelor două subpuncte, plecați de la următorul script:

hijack.php

```php
<?php
        $url = parse_url('http://localhost/your/address/session.php');
 $host = $url['host'];
 $path = $url['path'];

 $fp = fsockopen($host, 80);
 // send the request headers:
 fputs($fp, "GET $path HTTP/1.1\r\n");
 fputs($fp, "Host: $host\r\n");
 fputs($fp, "Connection: close\r\n\r\n");

 $result = '';
 while(!feof($fp)) {
  // receive the results of the request
  $result .= fgets($fp, 128);
 }
 // close the socket connection:
 fclose($fp);

 // split the result header from the content
 $result = explode("\r\n\r\n", $result, 2);

 $header = isset($result[0]) ? $result[0] : '';
 $content = isset($result[1]) ? $result[1] : '';

 echo $header;
?>
```

## Bibliografie


- http://phpsec.org/projects/guide/
- http://en.wikipedia.org/wiki/Session_fixation