WFC 02 – solutie oficiala pentru proba rapida

Avădănei Andrei, cu sprijinul lui Popescu Petre Alexandru - SirGod

Nota :

- fiecare vulnerabilitate are punctajul aferent
- sugerarea unei metode de rezolvare a problemei va duce la obtinerea punctajului maxim pentru respectiva vulnerabilitate, neindeplinirea acestei cerinte ducand la obtinerea a ½ din punctajul aferent problemei.
- Structurarea si modul de prezentare poate obtine intre 0 si 10 de puncte
- la aceasta proba solutia unui participant poate obtine maxim 650 de puncte

Programe/Setari/Diverse utilizate:

- WAMP Server (PHP, MySQL, phpMyAdmin, Apache)
- Notepad++
- Havij SQL Injection Tool
- Acunetix Vulnerability Scanner
- W3af (web application attack and audit framework)
- Functia PHP safe_input creata de Popescu Ionut Gabriel
- Platforma: Windows 7, Apache 2.2.17, PHP 5.2.5, MySQL 5.5.8, phpMyAdmin 3.3.19
- php.ini settings:

```
register_globals = on
allow_url_include = on
allow_url_fopen = on
display_errors = on
expose_php = on
display_startup_errors = on
log_errors = on
track_errors = on
report_memleaks = on
```

Functia safe_input:

```php
function safe_input($variable)
{
   $secured=str_replace('"',"&#34;",$variable);
   $secured=str_replace("'","&#39;",$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;
}
```

Functie care transforma toate caracterele speciale folosite in atacuri in entitati HTML. Functie folosita pentru a securiza inputul utilizatorului.

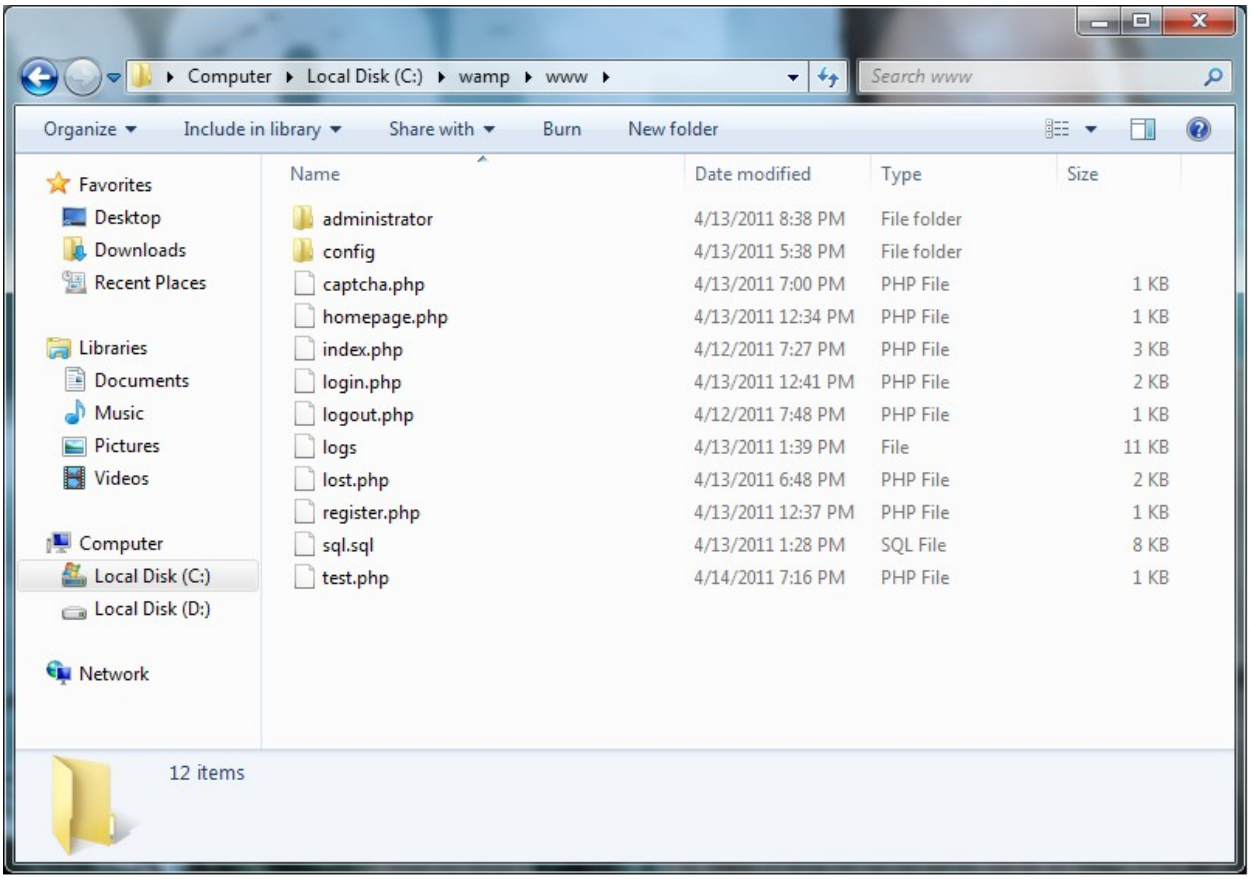
1) Descarc WAMP Server. Instalez. Descarc arhiva primita pe e-mail, selectez “proba rapida”. Copiez fisierele in folderul “www” din WAMP. Creez o baza de date pentru aplicatie. Rulez scriptul SQL. Tabelele si datele sunt introduse. Configurez

conexiunea la baza de date. Setez conditiile propice in php.ini pentru auditul de securitate.

2) Deschid toate fisierele si incep sa le analizez. Fiind obisnuit cu asta, greselile imi sar in ochi (variabile ale caror valoari poate fi manipulate de catre utilizator: variabile afisate nesatinizate, variabile incluse nesatinizate, action formuri fara tokenuri de protectie impotriva CSRF, variabile nesatinizate folosite cu baza de date, etc.).

3) Dupa ce termin auditul manual trec pe modul automat, folosind Acunetix si w3af. Dupa cum ma asteptam, nimic nou, doar o parte din vulnerabilitatile pe care le am descoperit manual (SQL Injection, XSS, RFI). Doar niste confirmari.

4) Am hotarat ca cel mai bine ar fi sa iau fiecare fisier in parte.



Incep cu directorul administrator.

5) administrator/editor.php **– 10pct**

Vulnerabilitatea nr.1: Admin Panel Access. Orice utilizator poate accesa panoul de administrator fara nicio restrictie, doar stiind calea catre fisier.

Fix: 1) .htaccess in folderul respectiv (restrictionare pe IP)

2) .htpassword (username+parola)

3) sistem de logare administrator (username si parola) sau acces pe baza de parola (PHP + MySQL)

Vulnerabilitatea nr.2: Local File Alteration. Orice utilizator poate altera fisiere deja existente dupa bunul plac (poate introduce cod PHP malitios in fisiere pentru a obtine shell). **- 10 pct**

```php
if(isset($_POST) && sizeof($_POST) == 3)
{
    if(file_exists($_POST['fname']))
    {
        $fh = fopen($_POST['fname'],'w');
        fwrite($fh, $_POST['fdata'],strlen($_POST['fdata']));
        fclose($fh);
        echo 'Successfully saved. <br />';
    }
}
```

Exemplu:

$fname= ../register.php

$fdata= <?php system($_GET['command']); ?>

Fix: 1) .htaccess in folderul respectiv (restrictionare pe IP)

2) .htpassword (username+parola)

3) sistem de logare administrator (username si parola) sau acces pe baza de parola (PHP + MySQL)

4) restrictionarea modificarii fisierelor din alte directoare decat cele permise (nu permitem caracterele speciale- functia safe_input)

```php
$alloweddir="../images/";


if(isset($_POST) && sizeof($_POST) == 3)
{
    if(file_exists($alloweddir.safe_input($_POST['fname'])))
    {
        $fh = fopen($_POST['fname'],'w');
        fwrite($fh, $_POST['fdata'],strlen($_POST['fdata']));
        fclose($fh);
        echo 'Successfully saved. <br />';
    }
}
```

6) Trecem la folderul config. Fisiere de configurare, de obicei nimic interesant. De obicei.

7) config/config.php

Vulnerabilitatea nr.3: Full Path Disclosure. Pentru ca codul este scris gresit, primim o eroare care ne dezvaluie calea intrega catre fisierul respectiv. **- 5pct**

```php
include_once('config/mysqlQuery.php');
include_once('config/users.php');
```

Fiind deja in folderul config incluziunea nu are loc si scriptul ne transmite eroarea.

Warning: include_once(config/mysqlQuery.php) [function.include-once]: failed to open stream: No such file or directory in C:\wamp\www\config\config.php on line *3*

Warning: include_once() [function.include]: Failed opening 'config/mysqlQuery.php' for inclusion (include_path='.;C:\php\pear') in C:\wamp\www\config\config.php on line *3*

Warning: include_once(config/users.php) [function.include-once]: failed to open stream: No such file or directory in C:\wamp\www\config\config.php on line *4*

Warning: include_once() [function.include]: Failed opening 'config/users.php' for inclusion (include_path='.;C:\php\pear') in C:\wamp\www\config\config.php on line *4*

Fatal error: Class 'MySQLQuery' not found in C:\wamp\www\config\config.php on line *5*

Fix: 1) .htaccess in folderul respectiv (deny from all)

2) includerea corecta a fisielor, fara directorul config

```php
include_once('mysqlQuery.php');
include_once('users.php');
```

3) folosirea operatorului @ (silent)

4) display_errors = off in php.ini

5) error_reporting(0);

8) Fisierul mysqlQuery.php nu are nicio problema, iar fisierul users.php va fi analizat mai tarziu, el continand toate functiile si nefiind exploatabil in mod direct.

9) captcha.php

Vulnerabilitatea nr.4: Denial of Service. Putem seta valori de ordinul miilor variabilelor width si height. Facand asta, in repetate randuri (sute de requesturi pentru imagini de 3000x3000), poate duce la consecinte de severitate medie, afectand serverul (supraincarcarea resurselor). **- 30 pct**

```
if(is_numeric(@$_GET['width'])) $w=$_GET['width'];else $w=160;
if(is_numeric(@$_GET['height'])) $h=$_GET['height'];else $h=25;
```

Fix: 1) impunerea limitei de valoare

```
if(is_numeric(@$_GET['width']) && $_GET['width'] < 200) $w=$_GET['width'];else $w=160;
if(is_numeric(@$_GET['height']) && $_GET['height'] < 200) $h=$_GET['height'];else $h=25;
```

2) variabile needitabile

```
$w=160;
$h=25;
```

Vulnerabilitatea nr.5: Captcha Bypass. Scriptul scrie valoarea captcha-ului intr-o variabila in cookie. Variabila scaptcha. Un atacator poate citi valoarea variabilei din cookie apoi o poate folosi pentru a trece de captcha. **- 20 pct**

Fix: 1) salvare valoare in sesiune, unde atacatorul nu are acces

```
$_SESSION['cvalue'] = $scaptcha;
```

apoi verificarea se va face din sesiune

```
if($_POST['captcha'] == $_SESSION['cvalue'])
```

10) homepage.php

Vulnerabilitatea nr.6: SQL Injection. Fisierul apeleaza functia isLogged() din config/users.php care este vulnerabila. Atacatorul poate injecta cod SQL in variabilele vpi_userID si vpi_userSession (prezente in cookie).

Prezentarea functiei plus explicatiile aferente si fixarea problemei vor fi mai spre sfarsit, pentru a nu scrie acelasi lucru de mai multe ori.

11) index.php

```
<title><?php if(@$_GET['page']=='home') echo 'P-Wannabe.com'; else echo urldecode($_SERVER['REQUEST_URI']);?></title>
```

Vulnerabilitatea nr.7: Cross-Site Scripting. Scriptul verifica valoarea primita de catre variabila page iar in cazul in care nu o primeste afiseaza (ca titlu) URL-ul cerut, care trece prin urldecode. Un request cu codul JavaScript urlcoded si codul JavaScript va fi executat cu succes. **- 20 pct**

http://127.0.0.1/?page=lost&sirgod=%3C/title%3E%3Cscript%3Ealert(10)%3C/script%3E

Fix: 1) folosirea functiei safe_input

```
<title><?php if(@$_GET['page']=='home') echo 'P-Wannabe.com'; else echo safe_input(urldecode($_SERVER['REQUEST_URI']));?></title>
```

2) folosirea functiei PHP htmlentities sau htmlspecialchars

```
<?php if(@$_GET['page']=='home') echo 'P-Wannabe.com'; else echo htmlentities(urldecode($_SERVER['REQUEST_URI']), ENT_QUOTES);?>
```

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Vulnerabilitatea nr.8: Remote/Local File Inclusion. Dupa cum putem observa mai jos, in cazul in care valoarea variabilei page nu este setata, valoarea ei va fi “home”. Mai jos putem observa ca ea este inclusa fara nicio protectie in prealabil. **- 20pct**

```php
include_once('config/config.php');
if(!defined('T_SERVER')) die();
if(isset($_GET['page'])) $page = $_GET['page']; else $page = 'home';
switch($page)
{
    case '':
    case 'home':
        include_once('homepage.php');
    break;
    case 'login':
        include_once('login.php');
    break;
    case 'logout':
        include_once('logout.php');
    break;
    case 'lost':
        include_once('lost.php');
    break;
    case 'register':
        include_once('register.php');
    break;
    default:
        include_once($page.'.php');
    break;
}
```

Un atacator poate seta valoarea variabilei un URL continand cod PHP malitios care va fi executat la incarcarea paginii. Este nevoie de un nullbyte (%00) sau semnul intrebarii (?) pentru a scapa de .php-ul care se adauga dupa valoare.

Fix: 1) folosirea functiei safe_input

2) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

```php
if(isset($_GET['page'])) {

if(!@eregi('[^a-z0-9_]', $page))

$page = $_GET['page'];

else $page = 'home';

switch($page)
{
    case '':
    case 'home':
        include_once('homepage.php');
    break;
    case 'login':
        include_once('login.php');
    break;
    case 'logout':
        include_once('logout.php');
    break;
    case 'lost':
        include_once('lost.php');
    break;
    case 'register':
        include_once('register.php');
    break;
    default:
        include_once($page.'.php');
    break;
}

}
```

Vulnerabilitatea nr.9: Full Path Disclosure. Un atacator poate seta variabila page primita prin $_GET ca array si poate provoca o eroare care dezvaluie intreaga cale de pe server catre fisierul respectiv. Exemplu: *http://127.0.0.1/?page[]=home* ***– 10 pct***

Warning: include_once(Array.php) [function.include-once]: failed to open stream: No such file or directory in C:\wamp\www\index.php on line *58*

Warning: include_once() [function.include]: Failed opening 'Array.php' for inclusion (include_path='.;C:\php\pear') in C:\wamp\www\index.php on line *58*

Fix: 1) includerea paginilor folosind operatorul @ (silent)

```php
switch($page)
{
    case '':
    case 'home':
        @include_once('homepage.php');
    break;
    case 'login':
        @include_once('login.php');
    break;
    case 'logout':
        @include_once('logout.php');
    break;
    case 'lost':
        @include_once('lost.php');
    break;
    case 'register':
        @include_once('register.php');
    break;
    default:
        @include_once($page.'.php');
    break;
}
```

2) display_errors = off in php.ini

12) login.php

Vulnerabilitatea nr.10: SQL Injection. Fisierul apeleaza functia isLogged() din config/users.php care este vulnerabila. Atacatorul poate injecta cod SQL in variabilele vpi_userID si vpi_userSession care se afla in cookie.

Prezentarea functiei plus explicatiile aferente si fixarea problemei vor fi mai spre sfarsit, pentru a nu scrie acelasi lucru de mai multe ori.

Vulnerabilitatea nr.11: SQL Injection. Fisierul apeleaza functia doLogin() din config/users.php care este vulnerabila. Atacatorul poate injecta cod SQL in variabilele username si password primite prin $_POST.

Prezentarea functiei plus explicatiile aferente si fixarea problemei vor fi mai spre sfarsit, pentru a nu scrie acelasi lucru de mai multe ori.

Vulnerabilitatea nr.12: Login Bypass – SQL Injection. Fisierul apeleaza functia doLogin() din config/users.php care este vulnerabila. Atacatorul poate injecta cod SQL in variabilele username si password intr-un anume fel in care poate trece de faza de logare.

Prezentarea functiei plus explicatiile aferente si fixarea problemei vor fi mai spre sfarsit, pentru a nu scrie acelasi lucru de mai multe ori.

Vulnerabilitatea nr.13: Cross-Site Scripting. Atacatorul poate introduce cod JavaScript, inputul utilizatorului nefiind satinizat. Ambele variabile, username si password, sunt vulnerabile. Exemplu: *"><script>alert(1)</script>* ***- 10pct***

```
Username : <input type="text" value="<?php echo @$_POST['username']; ?>" name="username" /><br />
Password : <input type="password" value="<?php echo @$_POST['password']; ?>" name="password" /><br />
```

Fix: 1) folosirea functiei safe_input

```
Username : <input type="text" value="<?php echo @safe_input($_POST['username']); ?>" name="username" /><br />
Password : <input type="password" value="<?php echo @safe_input($_POST['password']); ?>" name="password" /><br />
```

2) folosirea functiei PHP htmlentities sau htmlspecialchars

```
Username : <input type="text" value="<?php echo @htmlentities($_POST['username'], ENT_QUOTES); ?>" name="username" /><br />
Password : <input type="password" value="<?php echo @htmlentities($_POST['password'], ENT_QUOTES); ?>" name="password" /><br />
```

Vulnerabilitatea nr.14: Login Bruteforce. Deoarece formularul de logare nu este protejat nici de captcha, nici de un numar limitat de logari, un atacator poate face bruteforce pentru afla parola unui utilizator. **- 30 pct**

Fix: 1) folosire captcha la logare

2) limitarea numarului de logari esuate permise

Vulnerabilitatea nr. 14B: Security through obscurity. Este foarte util sa oferim cat mai putine informatii atacatorului cu privire la mesajele introduse de el. Spre exemplu, el ar putea afla adresa noastra de utilizator doar urmarind mesajele de eroare. Dupa ce a aflat acest lucru, nu ii ramane decat sa se axeze pe ghicirea parolei. **- 15pct**

```php
setcookie('tries', 1, time()+3600);

    if(isset($_POST) && sizeof($_POST) == 2)
{

    if($_COOKIE['tries'] <3) {

        $user   = $_POST['username'];
        $passwd = $_POST['password'];

        $state  = $users->doLogin($user, $passwd, true);

        switch($state)
        {
            case  1:echo '<META HTTP-EQUIV=Refresh CONTENT="0; URL=?page=home">';break;
            case -1:echo 'Invalid password.';break;
            case -2:echo 'Invalid username.'; setcookie('tries', $_COOKIE['tries'] +1); break;
            case -3:echo 'User logged.';break;
        }
        echo '<br />';
    }

    else die('Max login attempts reached.');

}
```

13) logout.php

Vulnerabilitatea nr.15: Cross-Site Request Forgery. La accesarea directa a paginii utilizatorul este delogat automat. Fara confirmare, fara token. Un atacator poate face un hyperlink imagine cu linkul si atunci cand utilizatorului logat ii este incarcata pagina, el va fi automat delogat de pe site. **- 30pct**

Fix: 1) folosire token (la logare sa se salveze un token random in sesiune care sa fie folosit pentru delogare, iar la cererea de delogare el sa fie cerut, in caz contrar sa nu se efectueze delogarea)

```php
public function doLogout()
{
    if($this->logged == TRUE && $_SESSION['logout_token'] = $token)
    {
       $this->db->query("DELETE FROM `sessions` WHERE session_id = '".$this->user_c."' AND user_id = '".$this->user_id."'");
       setcookie('vpi_userID'     ,'', $this->time - 10);
       setcookie('vpi_userSession','', $this->time - 10);

       $_SESSION['vpi_logged'] = FALSE;
       $this->session      = FALSE;
       $this->user_id      = -1;
       $this->user_c       = "";
       $this->logged       = FALSE;

       return TRUE;
    }
    return FALSE;
}
```

Vulnerabilitatea nr.16: SQL Injection. Fisierul apeleaza functia doLogout() din config/users.php care este vulnerabila. Atacatorul poate injecta cod SQL in variabilele vpi_userID si vpi_userSession care se afla in cookie.

Prezentarea functiei plus explicatiile aferente si fixarea problemei vor fi mai spre sfarsit, pentru a nu scrie acelasi lucru de mai multe ori.

14) lost.php

Vulnerabilitatea nr.17: SQL Injection. Variabila username trimisa prin $_POST este folosita nesatinizata, ceea ce permite unui atacator sa injecteze cod SQL malitios. **- 20pct**

```
$db->query("SELECT * FROM `users` WHERE username_clean = '".$_POST['username']."' LIMIT 1");
```

Fix: 1) folosirea functiei safe_input

```
$db->query("SELECT * FROM `users` WHERE username_clean = '".safe_input($_POST['username'])."' LIMIT 1");
```

2) folosirea functiei mysql_real_escape_string

```
$db->query("SELECT * FROM `users` WHERE username_clean = '".mysql_real_escape_string($_POST['username'])."' LIMIT 1");
```

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Vulnerabilitatea nr.18: SQL Injection. Fisierul apeleaza functia isLogged() din config/users.php care este vulnerabila. Atacatorul poate injecta cod SQL in variabilele vpi_userID si vpi_userSession care se afla in cookie.

Prezentarea functiei plus explicatiile aferente si fixarea problemei vor fi mai spre sfarsit, pentru a nu scrie acelasi lucru de mai multe ori.

Vulnerabilitatea nr.19: Password Reset Spree. Un “atacator” poate reseta parola administratorului incontinuu (in cazul in care stie username-ul si adresa de email + captcha bypass) deoarece nu exista o limita maxima admisa de resetari, facandu-i acelui utilizator logarea imposibila. **- 10pct**

Fix: 1) captcha cu salvare valoare in sesiune

```
$_SESSION['cvalue'] = $scaptcha;
```

```
if($_POST['captcha'] == $_SESSION['cvalue'])
```

2) limita de resetare per cont

3) o alta resetare sa nu fie posibila fara confirmarea prin e-mail

Vulnerabilitatea nr.20: Predictible Password. Un atacator poate reseta parola unui utilizator, apoi urmand sa faca bruteforce la campul de logare folosindu-se de problema scriptului: **- 50pct**

```
$npasswd = md5(rand(1,99999));
```

Sunt 99999 de parole posibile (la o rata de 10 parole pe secunda, va dura aproximativ 3 ore). Atacatorul poate face un dictionar cu toate parolele posibile si apoi poate face bruteforce (scriptul permite).

Fix: 1) generarea parolei sa se faca dupa un algoritm mai complex (exemplu)

```
$npasswd = md5(time()).rand(1,99999).md5(time).md5(rand(1,99999));
```

Vulnerabilitatea nr. 21: Posibilitate de CR/LF Injection. Validarea adresei de e-mail se face in mod eronat. Acesta foloseste o functie ce cauta un subsir intr-un sir, dar avand cei doi parametri inversati. Acest lucru ar putea duce la aflarea parolei abuzand de injectiile de tip CR/LF sau, de ce nu, la resetarea in masa a tuturor conturilor din baza de date in cazul in care programatorul va decide sa implementeze si acest lucru. (se observa ca programatorul nu a reusit sa termine scriptul). **- 50 pct**

```
$db->query("SELECT * FROM `users` WHERE username_clean = '".$_POST['username']."' LIMIT 1");
if($db->getNumRows() == 1)
{
   $npasswd = md5(rand(1,99999));
   $row = $db->fetch_array();

   if(stripos($_POST['email'], $row['email']) !== FALSE)
```

Fix: 1) if($_POST['email'] == $row['email'])

Vulnerabilitatea nr. 22: PHP Float DOS. O vulnerabilitate PHP <= 5.3.x a unor valori zecimale speciale ne permit suprasolicitarea serverului, prin consumarea tuturor resurselor CPU disponibile. Afisarea valorii 2.2250738585072011e-308 este un exemplu de astfel de valoare. Linia 38 este vulnerabila la aceasta problema de securitate. **- 50pct**

Fix: 1) Actualizarea versiunii PHP

2) Restrictionarea numerelor speciale :

2.2250738585072011e-308

.22250738585072011e-307

22.250738585072011e-309

22250738585072011e-324

02.2250738585072011e-308

2.22507385850720110e-308

2.2250738585072011e-0308

2.22507385850720111e-308

2.225073858507201123e-308

2.2250738585072011309 9e-308

2.22507385850720111000 01e-308

2.225073858507201112345678901234567890123456789012345678 9e-308

3) Limitarea valorilor acceptate. Spre exemplu, putem accepta valori <= x.999999

15) register.php

O linie de cod HTML, nimic interesant.

16) Vulnerabilitate generala – toate fisierele care includ config.php

Vulnerabilitatea nr.23: Full Path Disclosure. Atacatorul poate seta valoarea PHPSESSID-ului nula, in acest fel provocand o eroare care ne dezvaluie calea intreaga catre fisierul cu pricina. **- 10pct**

Warning: session_start() [function.session-start]: The session id is too long or contains illegal characters, valid characters are a-z, A-Z, 0-9 and '-,' in C:\wamp\www\config\config.php on line 2

Exemplu: executare *javascript:void(document.cookie="PHPSESSID=");* in campul pentru URL al browserului apoi se reincarca pagina.

Fix: 1) error_reporting(0); in config.php

```php
error_reporting(0);
session_start();
include_once('mysqlQuery.php');
include_once('users.php');
```

2) display_errors = off in php.ini

17) users.php

Functia __construct()

```php
public function __construct()
{
    global $db;
    if(!isset($_SESSION) || !is_array($_SESSION)) die("0 Session must be started!");
    $this->time        = time();
    $this->ip          = $_SERVER['REMOTE_ADDR'];
    $this->browser     = htmlspecialchars(substr($_SERVER['HTTP_USER_AGENT'],0,85));
    $this->session     = (bool)(isset($_SESSION['vpi_logged']) ? $_SESSION['vpi_logged'] : FALSE);
    $this->user_id     = (int)(isset($_COOKIE['vpi_userID']) ? $_COOKIE['vpi_userID'] : -1);
    $this->user_c      = htmlspecialchars(isset($_COOKIE['vpi_userSession']) ? $_COOKIE['vpi_userSession'] : "");
    $this->updateTime  = 60 * 60 * 24 * 1;

    $this->db          = $db;

    $this->doCheck();
    $this->logged = $this->isLogged();
}
```

In functia _construct avem toate variabilele necesare si prelucrarea lor. Desi la prima vedere variabilele vpi_userSession si vpi_userID par sigure, nu sunt. Fiind sceptic, le-am testat. Am creat un fisier test.php cu o conexiune la baza de date, am schimbat $_COOKIE cu $_GET (pentru a putea testa automat cu Havij), am modificat pseudo-variabila $this intr-una normala si am testat cu Havij daca campul este vulnerabil sau nu (mult mai eficient decat o testare manuala). Sa retinem, injectarea in variabilele acestea (vpi_userSession si vpi_userID- toate functiile) se va face in **cookie** (nu voi mai preciza ca injectia se face in cookie la fiecare functie in parte).

Forma originala vpi_userID:

```php
$this->user_id     = (int)(isset($_COOKIE['vpi_userID']) ? $_COOKIE['vpi_userID'] : -1);
```

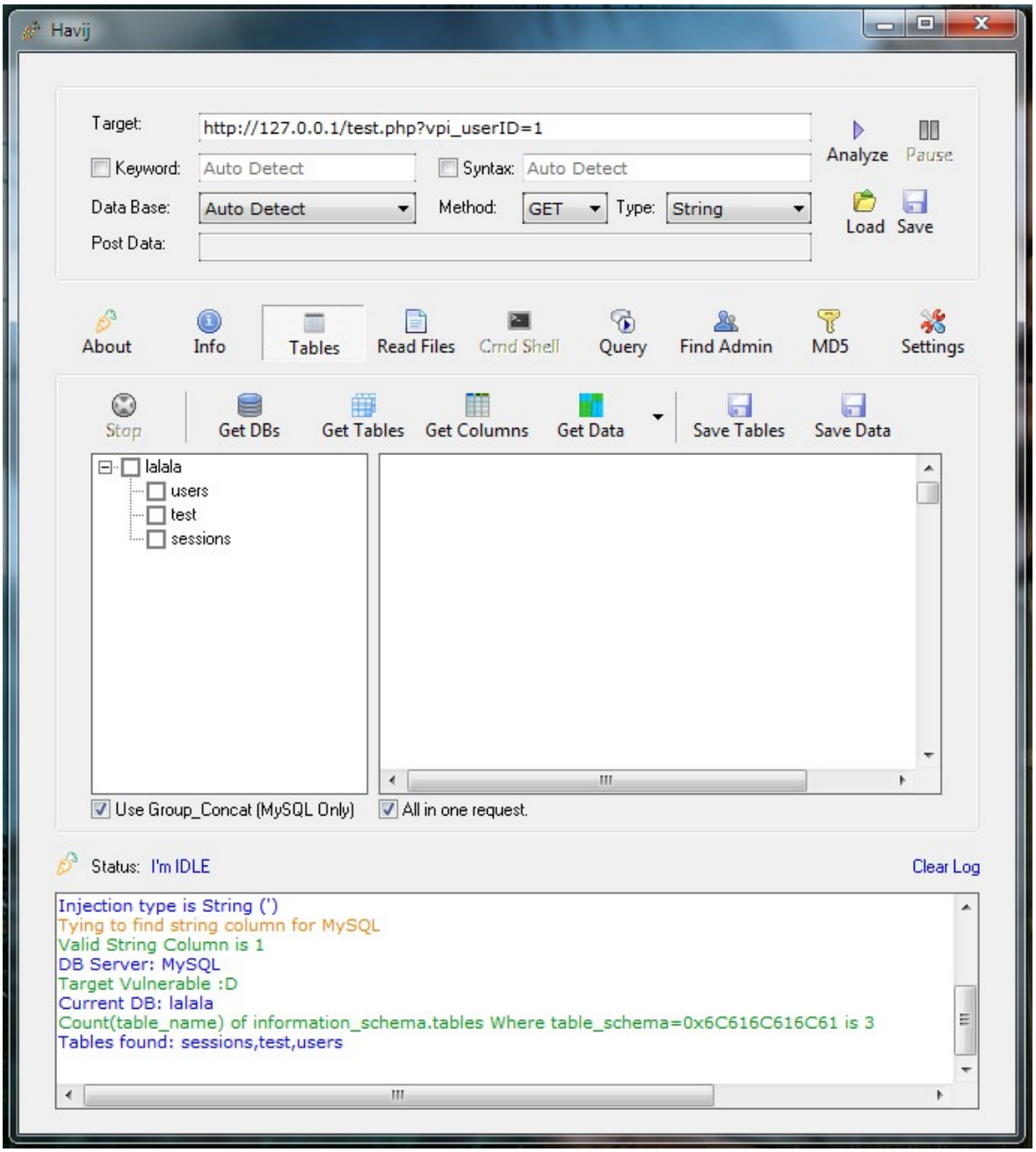
Fisierul test.php (testare vpi_userID)

```php
$AdresaBazaDate = "localhost";
$UtilizatorBazaDate = "root";
$ParolaBazaDate = "";
$NumeBazaDate = "lalala";
$conexiune = mysql_connect($AdresaBazaDate,$UtilizatorBazaDate,$ParolaBazaDate)
or die("Nu ma pot conecta la MySQL!");
mysql_select_db($NumeBazaDate,$conexiune) or die("Nu gasesc baza de date!");


$vpi_userID =  (int)(isset($_GET['vpi_userID'])) ? $_GET['vpi_userID'] : -1;

if($vpi_userID>=0) {

$cerereSQL = "SELECT password FROM `users` WHERE id='".$vpi_userID."' LIMIT 1";
$rezultat = mysql_query($cerereSQL);
while($rand = mysql_fetch_array($rezultat)) {
echo $rand['password'];
}
}
```



Dupa cum putem observa, variabila este nesatinizata, Havij a reusit sa faca injectia. Am verificat daca valoarea este mai mare sau egala decat 0 pentru ca functia isLogged() face asta.

Forma originala vpi_userSession:

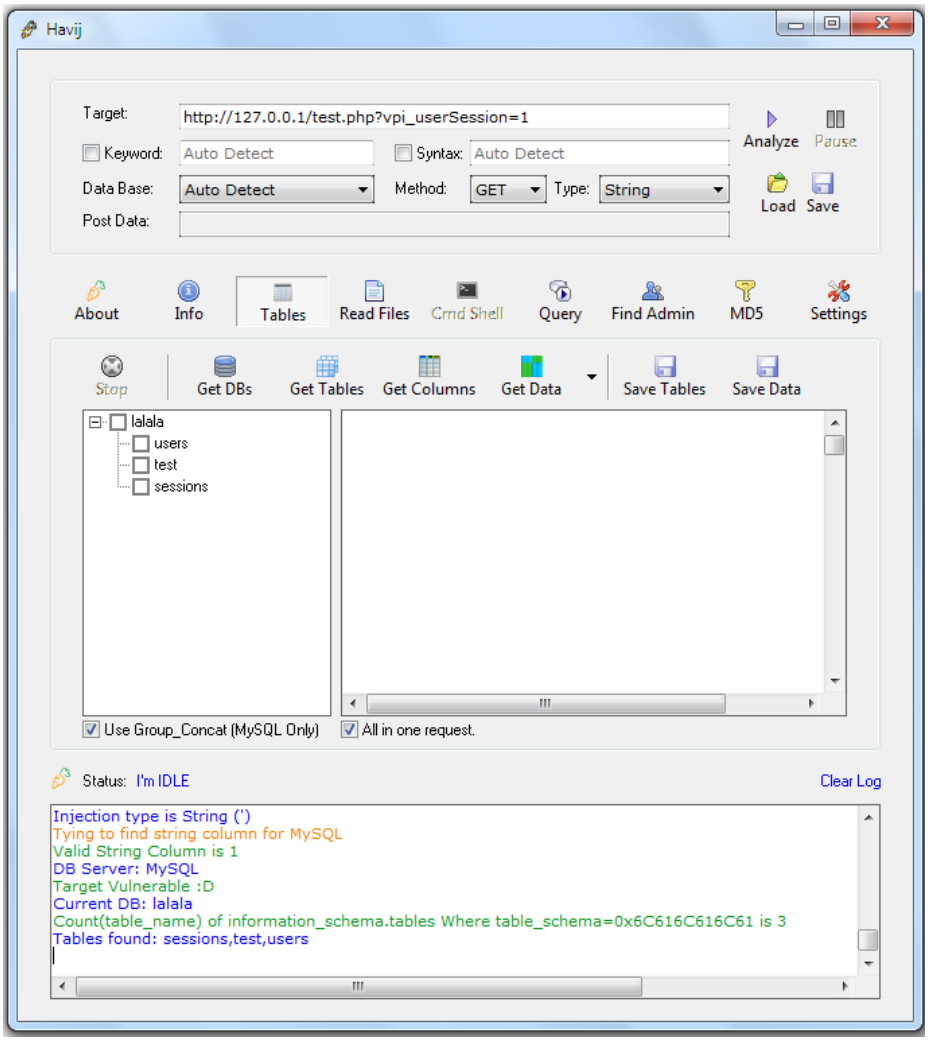
```
$this->user c      = htmlspecialchars(isset($ COOKIE['vpi userSession']) ? $ COOKIE['vpi userSession'] : "");
```

Fisierul test.php (testare vpi_userSession)

```php
$AdresaBazaDate = "localhost";
$UtilizatorBazaDate = "root";
$ParolaBazaDate = "";
$NumeBazaDate = "lalala";
$conexiune = mysql_connect($AdresaBazaDate,$UtilizatorBazaDate,$ParolaBazaDate)
or die("Nu ma pot conecta la MySQL!");
mysql_select_db($NumeBazaDate,$conexiune) or die("Nu gasesc baza de date!");


$vpi_userSession = htmlspecialchars(isset($_GET['vpi_userSession']) ? $_GET['vpi_userSession'] : "");

$cerereSQL = "SELECT password FROM `users` WHERE id='".$vpi_userSession."' LIMIT 1";
$rezultat = mysql_query($cerereSQL);
while($rand = mysql_fetch_array($rezultat)) {
echo $rand['password'];
}
```



Dupa cum putem observa, variabila este nesatinizata, Havij a reusit sa faca injectia.

Acum ca ne-am lamurit cum stau variabilele cu satinizarea sa trecem la analiza functiilor aflate in fisierul users.php.

Functia doCheck()

```
public function doCheck()
{
    if($this->user_c != "" && is_numeric($this->user_id))
    {
        $this->db->query("SELECT `id` FROM `users` WHERE id = '".$this->user_id."' LIMIT 1");
        if($this->db->getNumRows() == 1)
        {
            $this->db->query("SELECT * FROM `sessions` WHERE user_id = '".$this->user_id."' AND session_id = '".$this->user_c."' LIMIT 1");
            if($this->db->getNumRows() == 1)
            {
                $row = $this->db->fetch_array();
                if($this->browser == $row['browser'] && $this->ip == $row['ip'])
                {
                    if($this->time <= $row['time'])
                    {
                        $this->updateCookie();
                        return TRUE;
                    }
                    else
                    {
                        if((bool)$row['let_online'] == TRUE)
                        {
                            $this->updateCookie();
                            return TRUE;
                        }
                    }
                }
            }
        }
    }
    $this->deleteCookie();
    return FALSE;

}
```

Vulnerabilitatea nr.24 (generala): SQL Injection. Dupa cum putem observa, putem injecta cod SQL arbitrar doar in $this->user_c (variabila vpi_userSession) pentru ca $this->user_id (vpi_userID) trece prin functia is_numeric(). **- 10pct**

```
$this->db->query("SELECT * FROM `sessions` WHERE user_id = '".$this->user_id."' AND session_id = '".$this->user_c."' LIMIT 1");
```

Dupa cum putem vedea, variabila nu trece prin niciun filtru, permitandu-ne astfel sa injectam cod SQL (cookie).

Fix: 1) folosim functia safe_input()

```
$this->db->query("SELECT * FROM `sessions` WHERE user_id = '".$this->user_id."' AND session_id = '".safe_input($this->user_c).
"' LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Functia isLogged()

```
public function isLogged()
{
    if($this->session == TRUE)
    {
        if($this->user_c != "" && $this->user_id >= 0)
        {
            $this->db->query("SELECT * FROM `sessions` WHERE user_id = '".$this->user_id."' AND session_id = '".$this->user_c."' LIMIT 1"
            );
            if($this->db->getNumRows() == 1)
            {
                $row = $this->db->fetch_array();

                if($this->browser == $row['browser'] && $this->ip == $row['ip'] && $this->time <= $row['time'])
                {
                    return TRUE;
                }
            }
        }
    }
    return FALSE;
}
```

Vulnerabilitatea nr.25 (generala): SQL Injection. Dupa cum putem observa, putem injecta cod SQL arbitrar in $this->user_c (variabila vpi_userSession) cat si in $this->user_id (vpi_userID). **- 10pct**

Fix: 1) folosirea functiei safe_input

```
$this->db->query("SELECT * FROM `sessions` WHERE user_id = '".safe_input($this->user_id)."' AND session_id = '".safe_input(
$this->user_c)."' LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Functia doLogin()

```php
public function doLogin($user, $password, $online = FALSE)
{
    if($this->logged == FALSE)
    {
        $user     = htmlspecialchars(strtolower($user));

        $this->db->query("SELECT id FROM `users` WHERE username_clean = '".$user."' LIMIT 1");

        if($this->db->getNumRows() == 1)
        {
          $this->db->query("SELECT id,username,password FROM `users` WHERE username_clean = '".$user."' AND password = '".$password."' LIMIT
          1");

          if($this->db->getNumRows() == 1)
          {
            return $this->buildLogged($this->db->fetch_array(),$online); //logged
          }
          else
          {
            return -1; //invalid password
          }
        }
        else
        {
           return -2; //invalid  username
        }
    }
    else
    {
        return -3; //user logged
    }
}
```

Vulnerabilitatea nr.26 (generala): SQL Injection. Dupa cum putem observa, variabilele care trec prin functie sunt complet descoperite (in cazul nostru: username si password din login.php). Putem injecta cod SQL in oricare dintre cele doua variabile: user si password. Variabila user trece printr-un filtru obscur, care nu ajuta impotriva SQL Injection. **- 10pct**

Fix: 1) folosirea functiei safe_input

```php
$user     = htmlspecialchars(strtolower($user));

$this->db->query("SELECT id FROM `users` WHERE username_clean = '".safe_input($user)."' LIMIT 1");

if($this->db->getNumRows() == 1)
{
  $this->db->query("SELECT id,username,password FROM `users` WHERE username_clean = '".safe_input($user)."' AND password = '".
  safe_input($password)."' LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Vulnerabilitatea nr.27 (generala): Login Bypass – SQL Injection. Pe langa faptul ca putem executa cod SQL arbitrar, modul in care a fost scris scriptul ne permite sa trece peste faza de logare cu ajutorul unor simple stringuri. Query-ul cu pricina:

```php
$this->db->query("SELECT id,username,password FROM `users` WHERE username_clean = '".$user."' AND password = '".$password."' LIMIT
1");
```

Dupa cum putem observa, faza de logare poate fi omisa prin simpla introducere a unui string. Exemplu:

$user = 'or'1337'='1337

$password = 'or'1337'='1337

Asta va face ca codul (query-ul) sa devina:

```
$this->db->query("SELECT id,username,password FROM `users` WHERE username_clean = ''or'1337'='1337' AND password =
''or'1337'='1337' LIMIT 1");
```

Conditie adevarata, login bypass. **- 20pct**

Fix: 1) folosirea functiei safe_input

```
$this->db->query("SELECT id FROM `users` WHERE username_clean = '".safe_input($user)."' LIMIT 1");

if($this->db->getNumRows() == 1)
{
  $this->db->query("SELECT id,username,password FROM `users` WHERE username_clean = '".safe_input($user)."' AND password = '".
  safe input($password)."' LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Functia doRegister()

```
public function doRegister($username,$password,$email,$name,$about)
{
    $username       = htmlspecialchars($username);
    $username_clean = strtolower($username);
    $name           = htmlspecialchars($name);
    $email          = strtolower($email);
    $about          = htmlspecialchars($about);

    if($this->openreg == FALSE)             { return -8; } //registration are closed
    if(strlen($username) < 5)               { return -1; } //invalid username length
    if(strlen($password) < 5)               { return -2; } //invalid password length
    if($this->checkMail($email) == FALSE)   { return -3; } //invalid mail
    if(strlen($name) < 5)                   { return -4; } //invalid name

    if($this->logged == FALSE)
    {
        $this->db->query("SELECT id FROM `users` WHERE username_clean = '".$username_clean."' LIMIT 1");
        if($this->db->getNumRows() == 1)    { return -5; } //username allready exists

        $this->db->query("SELECT id FROM `users` WHERE email = '".$email."' LIMIT 1");
        if($this->db->getNumRows() == 1)    { return -6; } //email allready exists

        $this->db->query("INSERT INTO `users` (`username`,`username_clean`,`password`,`email`,`name`,`about`,`ip`,`date`)
                VALUES ('".$username."','".$username_clean."','".$password."','".$email."','".$name."','".$about."','".$this->ip.
                "','".gmdate( "D, d M Y H:i:s" )."')");
        return 1;     //user registered
    }
    else
    {
        return -7;   //user logged
    }
}
```

Vulnerabilitatea nr.28 (generala): SQL Injection. Variabila username_clean nu este satinizata. Variabila email trece prin functia checkMail si nu ne permite injectie de cod

SQL (sceptic fiind, am testat, decat sa stau sa analizez expresiile regulate care si asa nu sunt favoritele mele). **- 10pct**

Fix: 1) folosirea functiei safe_input

```
$this->db->query("SELECT id FROM `users` WHERE username_clean = '".safe_input($username_clean)."' LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Fisierul test.php (testare checkMail())

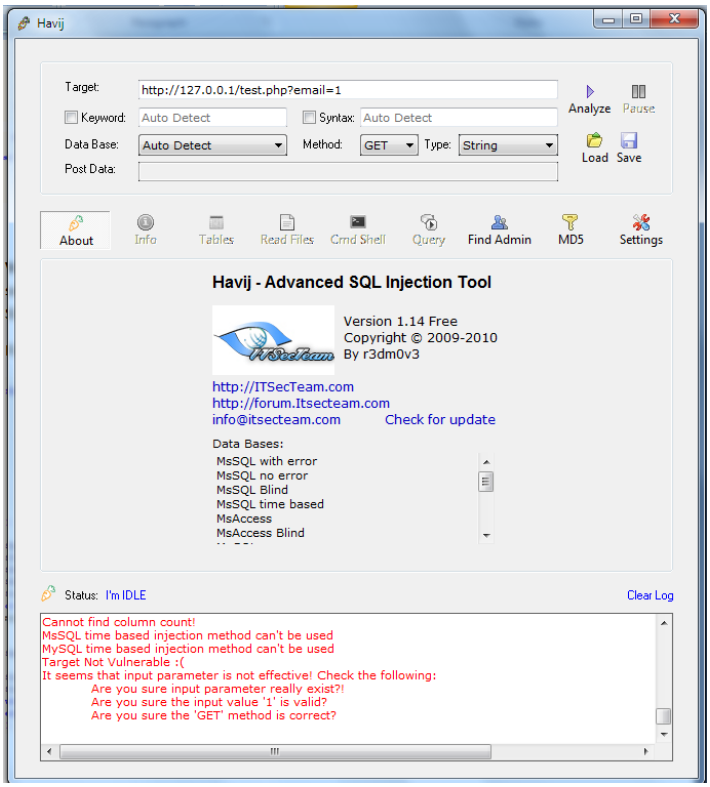
```
function checkMail($email)
    {
       if (!ereg("^[^@]{1,64}@[^@]{1,255}$", $email))
           return FALSE;

       $email_array = explode("@", $email);
       if(!preg_match('/(a-zA-Z0-9-_\.\)/',$email_array[0]))
           return FALSE;
       if(!preg_match('/(a-zA-Z0-9-_\.\)/',$email_array[1]))
           return FALSE;
       return TRUE;
}

$AdresaBazaDate = "localhost";
$UtilizatorBazaDate = "root";
$ParolaBazaDate = "";
$NumeBazaDate = "lalala";
$conexiune = mysql_connect($AdresaBazaDate,$UtilizatorBazaDate,$ParolaBazaDate)
or die("Nu ma pot conecta la MySQL!");
mysql_select_db($NumeBazaDate,$conexiune) or die("Nu gasesc baza de date!");


$email = checkMail($_GET['email']);

$cerereSQL = "SELECT password FROM `users` WHERE id='".$email."' LIMIT 1";
$rezultat = mysql_query($cerereSQL);
while($rand = mysql_fetch_array($rezultat)) {
echo $rand['password'];
}
```



Dupa cum putem observa, Havij nu a putut face injectia, ceea ce ne indreapta spre a crede ca nu putem executa cod SQL prin intermediul variabilei email.

Functia doUpdate()

```
public function doUpdate($password,$email,$name,$about,$npassword)
{
        $name           = htmlspecialchars($name);
        $email          = strtolower($email);
        $about          = htmlspecialchars($about);

        $username       = $this->getUsername();
        if(strlen($username) < 5)                       { return -1; } //invalid username length
        if($this->checkMail($email) == FALSE)           { return -2; } //invalid mail
        if(strlen($name) < 7)                           { return -3; } //invalid name 3 + 3 + 1

        if($this->logged == TRUE)
        {
            $this->db->query("SELECT id FROM `users` WHERE id = '".$this->user_id."' AND password = '".$password."' LIMIT 1");
               if($this->db->getNumRows() == 0)         { return -4; } //invalid username and password

            if($npassword != "" && strlen($npassword) < 5){ return -5; } //invalid new password

            if(strlen($npassword) >= 6)
            {
               $npassword      = md5(md5($npassword . $npassword) . md5($npassword));

               $this->db->query("UPDATE `users` SET password = '".$npassword."',email = '".$email."',name = '".$name."',about = '".$about."'
                                 WHERE id = '".$this->user_id."' AND password = '".$password."'");
            }
            else
            {
               $this->db->query("UPDATE `users` SET email = '".$email."',name = '".$name."',about = '".$about."' WHERE id = '".$this->user_id.
               "' AND password = '".$password."'");
            }

            return 1;
        }
        else
        {
            return -6;
        }
}
```

Vulnerabilitatea nr.29 (generala): SQL Injection. Dupa cum putem vedea, variabila password si $this->user_id (vpi_userID) sunt injectabile. Nicio protectie. **-10pct**

```
$this->db->query("SELECT id FROM `users` WHERE id = '".$this->user_id."' AND password = '".$password."' LIMIT 1");
```

Fix: 1) folosirea functiei safe_input

```
$this->db->query("SELECT id FROM `users` WHERE id = '".safe_input($this->user_id)."' AND password = '".safe_input($password)."'
LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Functia getInformation()

```
public function getInformation()
{
    if($this->logged == TRUE)
    {
        $this->db->query("SELECT * FROM `users` WHERE id = {$this->user_id} LIMIT 1");
        if($this->db->getNumRows() == 0)  { return -1; }

        $arr = $this->db->fetch_array();
        return '{
                "validate" :   "1",
                "id"       :   "'.$arr['id'].'",
                "username" :   "'.$arr['username'].'",
                "email"    :   "'.$arr['email'].'",
                "name"     :   "'.$arr['name'].'",
                "about"    :   "'.str_replace(array("\r","\n"),array('\r','\n'),$arr['about']).'",
                "ip"       :   "'.$arr['ip'].'",
                "date"     :   "'.$arr['date'].'"
        }';
    }

    return '{ "validate": "0" }';
}
```

Vulnerabilitatea nr.30 (generala): SQL Injection. Dupa cum putem vedea, pseudo-variabila $this->user_id (vpi_userID) este inclusa in query fara niciun fel de protectie permitandu-ne executarea de cod SQL. **- 10pct**

```
$this->db->query("SELECT * FROM `users` WHERE id = {$this->user_id} LIMIT 1");
```

Fix: 1) folosirea functiei safe_input

```
$this->db->query("SELECT * FROM `users` WHERE id = '".safe_input($this->user_id)."' LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Functia getUsername()

```
public function getUsername()
{
    if($this->logged == TRUE)
    {
        $this->db->query("SELECT username FROM `users` WHERE id = '".$this->user_id."' LIMIT 1");
        if($this->db->getNumRows() >= 1)
        {
            $username = $this->db->fetch_array();
            return $username['username'];
        }
    }
    return FALSE;
}
```

Vulnerabilitatea nr.31 (generala): SQL Injection. Dupa cum putem vedea, pseudo-variabila $this->user_id (vpi_userID) este inclusa in query fara niciun fel de protectie permitandu-ne executarea de cod SQL. **- 10pct**

```
$this->db->query("SELECT username FROM `users` WHERE id = '".$this->user_id."' LIMIT 1");
```

Fix: 1) folosirea functiei safe_input

```
$this->db->query("SELECT username FROM `users` WHERE id = '".safe_input($this->user_id)."' LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Functia getOptionalRow()

```
public function getOptionalRow()
{
   if($this->logged == TRUE)
   {
       $this->db->query("SELECT optional FROM `users` WHERE id='".$this->user_id."' LIMIT 1");
       if($this->db->getNumRows() == 1)
       {
           $optional = $this->db->fetch_array();
           return $optional['optional'];
       }
   }
   return FALSE;
}
```

Vulnerabilitatea nr.32 (generala): SQL Injection. Dupa cum putem vedea, pseudo-variabila $this->user_id (vpi_userID) este inclusa in query fara niciun fel de protectie permitandu-ne executarea de cod SQL. **- 10pct**

Fix: 1) folosirea functiei safe_input

```
$this->db->query("SELECT optional FROM `users` WHERE id='".safe_input($this->user_id)."' LIMIT 1");
```

2) mysql_real_scape_string

3) expresii regulate (validare alfanumerica)- ***if(!eregi('[^a-z0-9_]', $var))***

Functia doLogout()

```
public function doLogout()
{
   if($this->logged == TRUE)
   {
      $this->db->query("DELETE FROM `sessions` WHERE session_id = '".$this->user_c."' AND user_id = '".$this->user_id."'");
      setcookie('vpi_userID'     ,'', $this->time - 10);
      setcookie('vpi_userSession','', $this->time - 10);

      $_SESSION['vpi_logged'] = FALSE;
      $this->session      = FALSE;
      $this->user_id      = -1;
      $this->user_c       = "";
      $this->logged       = FALSE;

      return TRUE;
   }
   return FALSE;
}
```

Vulnerabilitatea nr.33 (generala): SQL Injection. Dupa cum putem vedea, functia doLogoutse foloseste de pseudo-variabila $this->logged care face apel la functia isLogged, care este vulnerabila la SQL Injection. Fixare si detalii puteti gasi la analiza functiei isLogged, prezente mai sus. **- 10pct**

```
$this->logged = $this->isLogged();
```

Functia updateOptional()

```
public function updateOptional($data)
{
    if($this->logged == TRUE)
    {
        $this->db->query("UPDATE `users` SET optional = '".$data."' WHERE id='".$this->user_id."'");
        return TRUE;
    }
    return FALSE;
}
```

Vulnerabilitatea nr.34 (generala): SQL Injection. Dupa cum putem vedea, functia updateOptional se foloseste de pseudo-variabila $this->logged care face apel la functia isLogged, care este vulnerabila la SQL Injection. Fixare si detalii puteti gasi la analiza functiei isLogged, prezente mai sus. **- 10pct**

```
if($this->logged == TRUE)
```

Functia getUserID()

```
public function getUserID()
{
    if($this->logged == TRUE)
    {
        return $this->user_id;
    }
    return FALSE;
}
```

Vulnerabilitatea nr.35 (generala): SQL Injection. Dupa cum putem vedea, functia getUserID se foloseste de pseudo-variabila $this->logged care face apel la functia isLogged, care este vulnerabila la SQL Injection. Fixare si detalii puteti gasi la analiza functiei isLogged, prezente mai sus. **- 10pct**

```
if($this->logged == TRUE)
```

Functia getCookie()

```
public function getCookie()
{
    if($this->logged == TRUE)
    {
        return '".
            "user_id":"'.$this->user_id.'",
            "user_c":"'.$this->user_c.'"
            ."';
    }

    return FALSE;
}
```

Vulnerabilitatea nr.36 (generala): SQL Injection. Dupa cum putem vedea, functia getCookie se foloseste de pseudo-variabila $this->logged care face apel la functia isLogged, care este vulnerabila la SQL Injection. Fixare si detalii puteti gasi la analiza functiei isLogged, prezente mai sus. **- 10pct**

```
if($this->logged == TRUE)
```

Vulnerabilitatea nr. 37 (generala): Plain-text passwords. Parolele sunt stocate in baza de date in format text, politica gresita ce usureaza munca unui hacker atunci cand reuseste sa treaca de sistemele de securitate de baza. **- 20pct**

Vulnerabilitatea nr. 38 (generala): Guessable config. Datele de acces din fisierul config sunt usor de ghicit. Acestea ar trebui schimbate. **- 20pct**

18) Functiile vulnerabile au fost numerate din nou ca vulnerabilitati (pe langa fisierele care le apeleaza) pentru ca ele pot fi folosite oriunde altundeva in script, lasand o gaura de securitate. Setarile php.ini au fost special alese pentru a putea gasi si testa

orice vulnerabilitate, deoarece anumite vulnerabilitati au nevoie de conditii special (e.g. LFI/RFI de allow_url_include).

19) Analiza logurilor (logs) **- 50pct**

Dupa cum putem observa, atacatorul a scanat site-ul cu scanerul de vulnerabilitati Acunetix.

Loguri de scanare cu Acunetix (partiale, am pastrat doar cateva care mi s-au parut interesante):

74.120.12.135 - - [13/Apr/2011:09:55:48 +0300] "GET /acunetix-wvs-test-for-some-inexistent-file HTTP/1.1" 404 431 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Probabil folosit pentru injectia in loguri (testare)**

74.120.12.135 - - [13/Apr/2011:09:55:49 +0300] "GET /acunetix-wvs-test-for-some-inexistent-file-second-try HTTP/1.1" 404 436 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Probabil folosit pentru injectia in loguri (testare)**

77.247.181.165 - - [13/Apr/2011:09:55:57 +0300] "GET /inexistent_file_name.inexistent0123450987.cfm HTTP/1.1" 404 434 "-" "<script>alert(12345)</script>" **XSS**

178.202.59.102 - - [13/Apr/2011:09:56:13 +0300] "GET /_vti_pvt/authors.pwd HTTP/1.1" 404 422 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Sensitive file**

178.202.59.102 - - [13/Apr/2011:09:56:14 +0300] "GET /long_inexistent_path12345_/Null.htw?CiWebhitsfile=:&CiRestriction=b&CiHiliteType=full HTTP/1.1" 404 470 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Probabil folosit pentru injectia in loguri (testare)**

178.202.59.102 - - [13/Apr/2011:09:56:15 +0300] "GET /wp-content/plugins/wordpress-file-monitor/wordpress-file-monitor.php?ver=scan HTTP/1.1" 200 - "http://www.p-wannabe.com/" "Opera/7.54 (Windows NT 5.1; U) [en]" **Probabil sensitive file/ Control panel**

178.202.59.102 - - [13/Apr/2011:09:56:15 +0300] "GET /web-console/Invoker HTTP/1.1" 404 419 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Control panel**

178.202.59.102 - - [13/Apr/2011:09:56:17 +0300] "POST /_vti_bin/shtml.exe?_vti_rpc HTTP/1.1" 404 425 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Sensitive file**

178.202.59.102 - - [13/Apr/2011:09:56:18 +0300] "GET /_vti_inf.html HTTP/1.1" 404 413 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Sensitive file**

178.202.59.102 - - [13/Apr/2011:09:56:18 +0300] "GET /stronghold-info HTTP/1.1" 404 415 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Sensitive file**

178.202.59.102 - - [13/Apr/2011:09:56:19 +0300] "GET /tomcat-docs/appdev/sample/web/hello.jsp?test=<script>alert(12345)</script> HTTP/1.1" 404 467 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **XSS**

178.202.59.102 - - [13/Apr/2011:09:56:19 +0300] "GET /web-console/ HTTP/1.1" 404 413 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Control Panel**

178.202.59.102 - - [13/Apr/2011:09:56:22 +0300] "GET /jmx-console/HtmlAdaptor?action=inspectMBean&name=jboss.system:type%3DServerInfo HTTP/1.1" 404 467 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Control Panel**

178.202.59.102 - - [13/Apr/2011:09:56:23 +0300] "GET /..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5c/windows/win.ini HTTP/1.1" 404 425 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **LFI Windows**

178.202.59.102 - - [13/Apr/2011:09:56:25 +0300] "POST /_vti_bin/_vti_aut/author.dll HTTP/1.1" 404 426 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **Sensitive file**

178.202.59.102 - - [13/Apr/2011:09:56:33 +0300] "GET /sdk/../../../../../../../../../../../../etc/passwd HTTP/1.1" 404 420 "-" "Opera/7.54 (Windows NT 5.1; U) [en]" **LFI Linux**

Dupa cum putem observa, teste de LFI (windows & linux) au fost efectuate, teste XSS, teste pentru a gasi diverse aplicatii, panouri de administrare sau fisiere importante neprotejate.

20) Detalii despre atacator **- 10pct**

Adrese IP (probabil mascate: proxy/socks):

74.120.12.135 (US)

178.202.59.102 (Germany)

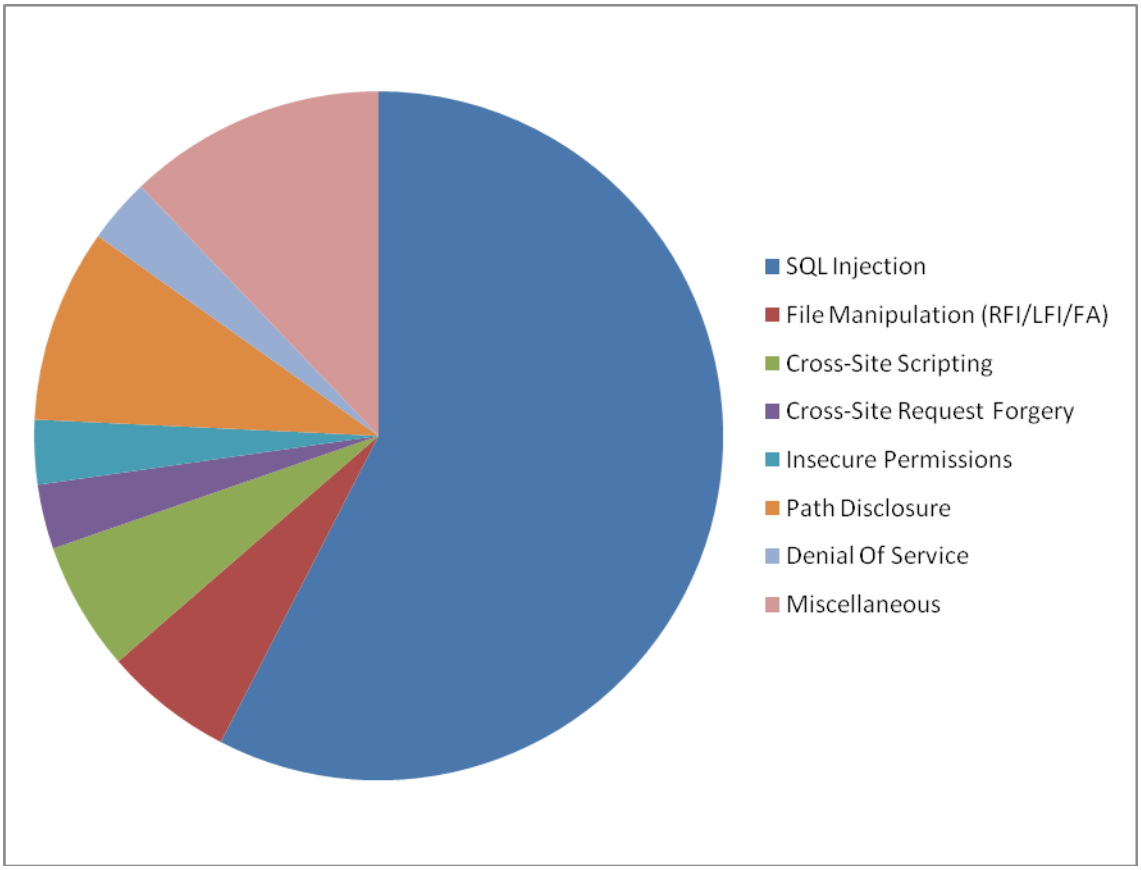
77.247.181.165 (Netherlands)

Sistem de operare: *Windows XP (Windows NT 5.1)*

Browser: *Opera/7.54 (probabil spoofed)*

21) Detalii generale despre script

Contine de la vulnerabilitati cu impact minim pana la vulnerabilitati cu impact maxim asupra securitatii website-ului cat si a serverului web.



Dupa cum putem vedea in acest grafic, SQL Injection este vulnerabilitatea cea mai raspandita in script, o vulnerabilitate cu grad de risc mediu, urmata de File Manipulation, o vulnerabilitate cu grad de risc ridicat.

Asta-i tot.