# Tvorba webových aplikací 1

## Přednáška 9:

# Bezpečnost PHP aplikací

## (Přednáška založená na prezentaci Iliu Alshanetskeho)

# Osnova přednášky

- Ošetření vstupu
- Útoky:
  - Cross-site scripting
  - Cross-Site Request Forgeries
  - SQL injection
  - Vzdálené spouštení kódu
- Chybová hlášení
- Soubory
- Bezpečnost session
- Hesla

# Bezpečnost

- Bezpečnost je veličnia, ne vlastnost
  - aplikace může být víc nebo méně bezpečná
  - aplikace nemůže být absolutně bezpečná
- Je potřebné poznat bezpečnostné hrozby a předcházet jim
  - při návrhu aplikace
  - při implementaci

- Nezneužívejte prosím informace z této přednášky :)

# Ošetření vstupu

- **Uživatelský vstup není důvěryhodný:**
  - Může být částečně ztracen nebo změněn cestou na server.
  - Může být chybně zadán uživatelem.
  - Může být záměrně zadán za účelem získat kontrolu nad aplikací.
- **Každý uživatelský vstup musí být před použitím validován.**

# Vstupní data

- Počínaje PHP 4.1 má PHP množinu super globalních proměnných pro přístup ke vstupním hodnotám:
  - $_GET – data z požadavků GET.
  - $_POST – data z požadavků POST.
  - $_COOKIE – cookie data.
  - $_FILES – uploadované soubory.
  - $_SERVER – serverová data
  - $_ENV – proměnné prostředí
  - $_REQUEST – kombinace GET/POST/COOKIE

# Register Globals

- Nejčastější zdroj bezpečnostních děr v PHP.
- Každý vstupní parametr je přístupný jako stejně pojmenovaná proměnná:
  - script.php?foo=bar >> $foo = “bar”;
  - Neznáme zdroj vstupu (cookie, get, post...)
  - Neinicializované proměnné mohou být “podstrčeny” uživatelem.

# Register Globals

```
if (authenticated_user()) {
   $authorized = true;
}
if ($authorized) {
   include '/highly/sensitive/data.php';
}
```

- `$authorized` není v skriptu inicializováno
- zaslaním GET požadavku

  http://example.com/script.php?authorized=1

  získá útočník přístup k citlivým datům

# Řešení pro Register Globals

- Vypnout `register_globals` v php.ini.
  - V základní konfiguraci od PHP 4.2.0
- Programovat s `error_reporting` nastaveným na `E_ALL`.
  - umožňuje vidět neinicializované proměnné.

# Další problémy s Register Globals

```
$var[] = “123”;
foreach ($var as $entry) {
   make_admin($entry);
}
```

- **Dotaz**

  **script.php?var[]=1&var[]=2**

  **vloží do pole  $var dvě hodnoty**
- **Vložení v PHP není možné detekovat.**

# $_REQUEST

- Pole $_REQUEST kombinuje vstupy z POST, GET a COOKIE
- **Může dojít ke kolizi hodnot... nepoužívat!**

# $_SERVER

- Even though the $_SERVER super-global is populated based on data supplied by the web-server it should not be trusted.
  - User may inject data via headers

    ```
    Host: <script> ...
    ```

  - Some parameters contain data based on user input

    ```
    REQUEST_URI, PATH_INFO, QUERY_STRING
    ```

  - Can be fakes

    Spoofed IP address via the use of anonymous proxies.

# Validace číselných vstupních dat

- Všechna vstupní data přicházejí do PHP jako řetězec (string).
- Číselný vstup by neměl být používán jako s řetězec
  - nebezpečí chyby aplikace nebo napadení
- Všechna číselná data by měla být validována:

```
// integer validation
if (!empty($_GET['id'])) {
        $id = (int) $_GET['id'];
} else
        $id = 0;
// floating point number validation
if (!empty($_GET['price'])) {
        $price = (float) $_GET['price'];
} else
        $price = 0.0;
```

# Validace řetězců

- PHP poskytuje funkce `ctype...` pro validaci řetězců:

```
if (!ctype_alnum($_GET['login'])) {
        echo "Only A-Za-z0-9 are allowed.";
}
if (!ctype_alpha($_GET['captcha'])) {
        echo "Only A-Za-z are allowed.";
}
if (!ctype_xdigit($_GET['color'])) {
        echo "Only hexadecimal values are allowed";
}
```

# Validace cesty

- Použít uživatelský vstup pro určení cesty k souboru může být nebezpečné...

```
http://example.com/script.php?path=../../etc/passwd
```

```
<?php
$fp = fopen("/home/dir/{$_GET['path']}", "r");
?>
```

# Validace cesty

- PHP funkce `basename()` odstraní z řetězce cesty všechno kromě poslední části cesty (jména souboru):

```
<?php
$_GET['path'] = basename($_GET['path']);

// only open a file if it exists.
if (file_exists("/home/dir/{$_GET['path']}")) {
   $fp = fopen("/home/dir/{$_GET['path']}", "r");
}
?>
```

# Validace cesty

- Lepší je však ukrýt názvy souborů a poskytnout uživateli jenom seznam kódů

```
// make white-list of templates
$tmpl = array();
foreach(glob("templates/*.tmpl") as $v) {
        $tmpl[md5($v)] = $v;
}
if (isset($tmpl[$_GET['path']]))
        $fp = fopen($tmpl[$_GET['path']], "r");

http://example.com/script.php?path=57fb06d7...
```

# magic_quotes_gpc

- PHP má zabudováno automatické ošetření speciálních znaků ve vstupu ( `', ", \, \0 (NULL)` )
- Nevýhody
  - Zpomaluje zpracování vstupu.
    - Ošetření číselných vstupů je možné dělat lépe přetypovaním.
    - Vyžaduje 2 násobek paměti pro vstupní data.
  - Na serveru může být vypnuté.
  - Velice obecné řešení (někdy musíme ošetřit i jiné znaky).
- Je dobré mít funkci, která, jestli je *magic_quotes* zapnuté, zruší lomítka a pak vlastní funkci pro ošetření vstupu

# Magic Quotes

```
if (get_magic_quotes_gpc()) { // check magic_quotes_gpc state
        function strip_quotes(&$var) {
                if (is_array($var)
                        array_walk($var, 'strip_quotes');
                else
                        $var = stripslashes($var);
        }

     // Handle GPC
        foreach (array('GET','POST','COOKIE') as $v)
                if (!empty(${"_".$v}))
                        array_walk(${"_".$v}, 'strip_quotes');

    // Original file names may contain escaped data as well
        if (!empty($_FILES))
                foreach ($_FILES as $k => $v) {
                  $_FILES[$k]['name'] = stripslashes($v['name']);
}
```

# Exploiting Code in Previous Slide

- While the code on the previous slide works, it can be trivially exploited, due to its usage of recursive functions!

```
<?php
$qry = str_repeat("[]", 1024);
$url = "http://site.com/script.php?a{$qry}=1";
file_get_contents($url);

// run up in memory usage, followed by a prompt
   crash
?>
```

# Odstranění vlivu magic_quotes

```
if (get_magic_quotes_gpc()) {
        $in = array(&$_GET, &$_POST, &$_COOKIE);
        while (list($k,$v) = each($in)) {
                foreach ($v as $key => $val) {
                        if (!is_array($val)) {
                                $in[$k][$key] = stripslashes($val);
                                continue;
                        }
                        $in[] =& $in[$k][$key];
                }
        }
        unset($in);
}
```

# *Cross Site Scripting* (XSS)

- Cross Site Scripting (XSS) je situace, kdy útočník vloží do stránky škodlivý HTML kód, který se pak zobrazí na stránce (např. ve fóru)
  - Znehodnocení stránky.
  - Převzetí session.
  - Krádeže hesla.
  - Sledování činnosti uživatelů.

# Prevence XSS

- Prevence spočívá ve filtrování zobrazovaného uživatelského vstupu následovnými funkcemi:
  - `htmlspecialchars()`
    - Převede zvláštní znaky na HTML entity (‘, “, <, >, &)
  - `htmlentities()`
    - Převést všechny použitelné znaky na HTML entity.
  - `strip_tags()`
    - Odstraní z řetězce HTML a PHP tagy.
  - `mysqli_real_escape_string()`
    - Ošetření speciálně pro použití v MySQL.

# Prevence XSS

```
$str = strip_tags($_POST['message']);
// encode any foreign & special chars
$str = htmlentities($str);
// maintain new lines, by converting them to <br />
echo nl2br($str);

// strip tags can be told to "keep" certain tags
$str = strip_tags($_POST['message'], '<b><p><i><u>');
$str = htmlentities($str);
echo nl2br($str);
```

- Povolení tagů v `strip_tags()` (druhý argument) je nebezpečné, protože atributy těchto tagů zůstanou bez kontroly.

# Problém povolení tagů

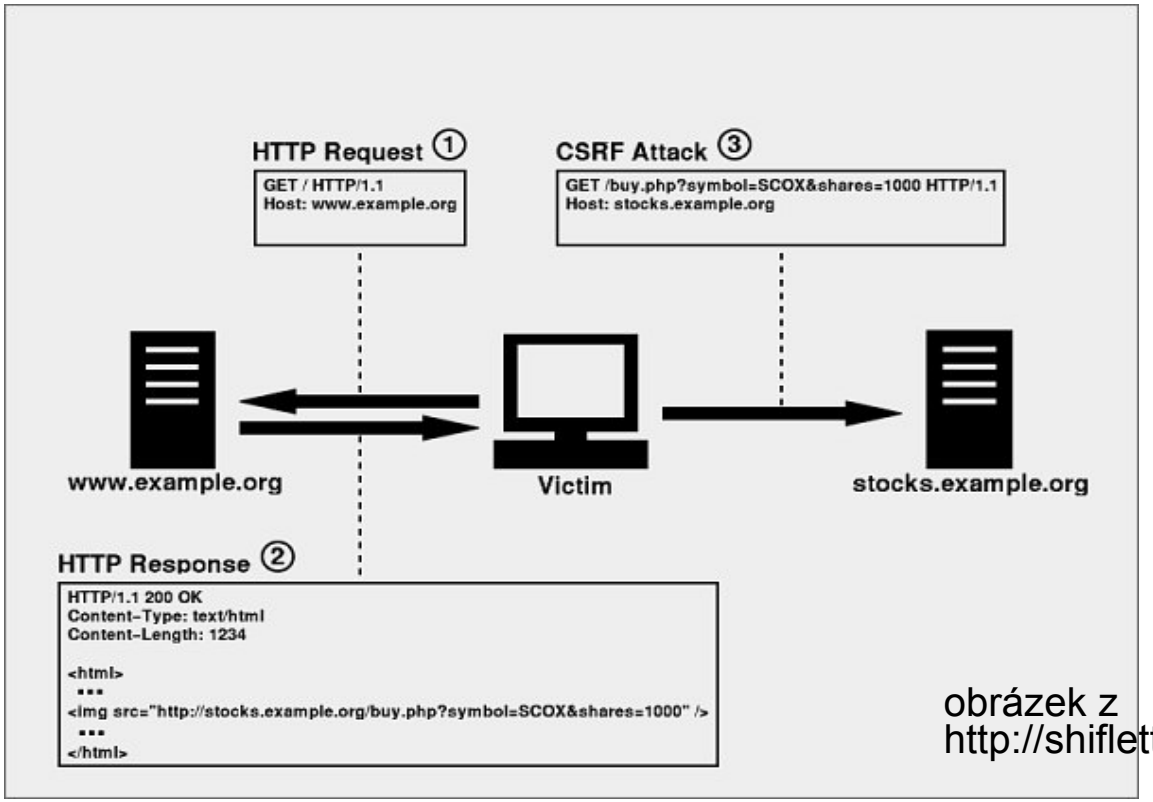
```
<b style="font-size: 500px">
TAKE UP ENTIRE SCREEN
</b>

<u onmouseover="alert('JavaScript is allowed');">
<b style="font-size: 500px">Lot's of text</b>
</u>

<p style="background:
  url(http://tracker.com/image.gif)">
Let's track users
</p>
```

# *Cross-Site Request Forgeries* (XSRF)

- Uživatel (oběť) navštíví stránku útočníka.
- Stránka bez vedomí uživatele vykoná akci na (cílovém) serveru, na který je uživatel legálně přihlášen.




obrázek z
http://shiflett.org/articles/foiling-cross-site-attacks

# Prevence XSRF

- Používat POST pro všechny akce.
- Vyžadujte potvrzení akce
  - a. potvrzení dodatečným kliknutím
  - b. potvrzení heslem (u důležitých akcí)
- *Anti-CSRF Token*
  - *kód vygenerován u každého zobrazení formuláře*
  - *kód se zobrazí u formuláře a zároveň uloží do SESSION*
  - *při přijetí požadavku s daty formuláře porovnáme $_POST['token'] a $_SESSION['token']*
  - *timeout může být nastaven pro platnost tokenu*

# *SQL Injection*

- *SQL injection* se podobá XSS.
- Nekontrolovaný uživatelský vstup je použit při vytváření SQL dotazu.
- Pomocí vloženého dodatečného SQL dotazu do vstupu může útočník:
  - smazat, vložit nebo měnit data
  - zapříčinit DoS
- Příklad:

  http://example.com/db.php?id=0;DELETE%20FROM%20users

# Prevence SQL injection: SQL escaping

- PHP API pro databáze má často speciální funkce pro úpravu vstupu:
  - MySQL
    - `mysql_escape_string()`
    - `mysqli_real_escape_string()`
  - PostgreSQL
    - `pg_escape_string()`
    - `pg_escape_bytea()`
  - SQLite
    - `sqlite_escape_string()`

# Escaping v praxi

```
// undo magic_quotes_gpc to avoid double escaping
if (get_magic_quotes_gpc()) {
  $_GET['name'] = stripslashes($_GET['name'];
  $_POST['binary'] = stripslashes($_GET['binary']);
}
```

```
$name = pg_escape_string($_GET['name']);
$binary = pg_escape_bytea($_POST['binary']);
```

```
pg_query($db, "INSERT INTO tbl (name,image)
       VALUES('{$name}', '{$image}')");
```

# Nedostatky SQL escaping-u

- Když do SQL dotazu vkládáme holý integer, escaping nefunguje
  - žádné speciální znaky na úpravu

```
http://example.com/db.php?id=0;DELETE%20FROM%20users
<?php
$id = sqlite_escape_string($_GET['id']);
// $id is still 0;DELETE FROM users

sqlite_query($db,
     "SELECT * FROM users WHERE id={$id}");
// Bye Bye user data...
?>
```

# *Prevence SQL injection: Prepared Statements*

- *Prepared statements* slouží k zabezpečení a optimalizaci vykonávaných dotazů.
- SQL “zkompiluje” dotaz a pak při každém vykonání jenom nahrazuje hodnoty přeměnných.
  - Vyšší výkon – jedno kompilování na dotaz.
  - Vyšší bezpečnost, vložená data nejsou považována za další dotaz.
  - Podpora v hlavních RDBMS:
    - MySQL od verze 4.1.
    - SQLite nemá podporu.

# Použití *Prepared Statements*

```
<?php
$data = "Here is some text to index";

pg_query($db, "PREPARE my_stmt (text) AS
       INSERT INTO search_idx (word) VALUES($1)");
foreach (explode(" ", $data) as $word) {
        // no is escaping needed
        pg_query($db, "EXECUTE my_stmt({$word})");
}

// de-allocte the prepared statement
pg_query($sb, "DEALLOCATE my_stmt");
?>
```

# Vzdálené spuštění kódu

- Remote code execution, Remote file inclusion...
- Útočník spustí prostřednictvím napadené webové aplikace vlastní kód:
  - načítaním vlastního souboru (lokálního, nebo ze vzdáleného serveru)
  - spuštěním eval() na nekontrolovaném vstupu
- Hrozí převzetí kompletní kontroly nad napadenou aplikací nebo celým serverem.

# Vzdálené spuštění kódu: ochrana

- Omezit povolených jmen souborů pro upload.
- Vkládat kód mohou jenom důvěryhodní uživatelé (ne stroje).
- Nepoužívat funkci *eval()* s neošetřeným vstupem.
- Nevkládat (include) nedůvěryhodné soubory.
- Ošetřovat (escape) všechny příkazy pro *shell*.

# Chybová hlášení

- Vypisování PHP chyb v produkčním nasazení:
  - obtěžuje uživatele
  - může prozradit důležité informace:
    - cesty k souborům a skriptům,
    - neinicializované proměnné,
    - parametry podávané funkcím.
- Vypnutí chybových hlášení znemožní nalezení chyb v kódu.
- Potřeba řešení...

# Chybová hlášení: jedno z řešení

- Vypnout vypisování chyb na obrazovku:

```
ini_set("display_errors", FALSE);
```

- Zapnout log chyb na disku:

```
ini_set("log_errors", TRUE);
```

- do souboru:

```
ini_set("error_log", "/var/log/php.log");
```

- nebo do *syslog*-u:

```
ini_set("error_log", "syslog");
```

- **To samé se dá udělat v *php.ini***

# Bezpečnost souborů

- PHP aplikace obvykle mají soubory obsahující:
  - konfigurace
  - knihovny pomocných funkcí
  - slovníky.

- Tyto soubory musí webserver číst, mohou být viditelné z webu.
- Je-li tomu tak, uživatelé je můžou stáhnout a číst.

# Zabezpečení souborů

- Do kořenového adresáře aplikace nedávejte žádné soubovy, které tam nemají být.
- Když soubor nemá žádný výstup (knihovna, konfigurace), dejte mu příponu PHP
  - to znamená, že ho pri pokusu o zobrazení z webu bude parsovat PHP parser
- Používejte .htaccess na řízení přístupu k souborům a adresářům

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

# Zabezpečení konfiguračních souborů

- Konfigurační soubory obsahují citlivá data.
- Přístup z webu řešily předchozí stránky.
- Soubory zůstávají čitelné pro uživatele uvnitř systému.
- Ideálně by měly být tyto soubory čitelné jenom pro jejich vlastníka.
- Řešení...

# Řešení #1

- Vyžaduje přístup k Apache.
- Nastavení připojení k DB mohou být určeny ini direktivami.
- Soubor s nastavením se pak nahraje do httpd.conf pomocí příkazu `Include`.

**mysql.cnf**

```
mysql.default_host=localhost
mysql.default_user=forum
mysql.default_password=secret
```

**httpd.conf**

```
<VirtualHost 1.2.3.4>
Include "/site_12/mysql.cnf"
</VirtualHost>
```

- Apache přistupuje k souboru jako "root", takže soubor může mít práva (0600).

# Řešení #2

- Vyžaduje přístup k Apache
- Ostatní nastavení mohou být uložena do proměnných serveru.

misc_config.cnf

```
SetEnv NNTP_LOGIN   "login"
SetEnv NNTP_PASS    "passwd"
SetEnv NNTP_SERVER "1.2.3.4"
```

httpd.conf

```
<VirtualHost 1.2.3.4>
Include "misc_config.cnf"
</VirtualHost>
```

```
echo $_SERVER['NNTP_LOGIN']; // login
echo $_SERVER['NNTP_PASS']; // passwd
echo $_SERVER['NNTP_SERVER']; // 1.2.3.4
```

# Hesla a bezpečnost

- Hesla musí být vždy uložena v DB zakódováně (sha1, md5)
  - mysql_query("SELECT * FROM uzivatele WHERE login = '$_POST[login]' AND heslo_sha1 = '" . sha1(stripslashes($_POST["heslo"])) . "'");
- Při změně hesla vždy požadujeme původní heslo.
  - mysql_query("UPDATE uzivatele SET heslo_sha1 = SHA1('$_POST[heslo]') WHERE login = '$_POST[login]' AND heslo_sha1 = SHA1('$_POST[heslo_puvodni]')");
- Přidání náhodného řetězce (*salt)* k heslu předcházíme jeho prozrazení kvůli duplicitním heslům:
  - *salt* je uložen jako řetězec s heslem u uživatele,
  - heslo je uloženo se *salt* a zakódováno
  - mysql_query("SELECT * FROM uzivatele WHERE login = '$_POST[login]' AND heslo_sha1 = SHA1(CONCAT('$_POST[heslo]', salt))");

# *Shared Hosting*

- Hosting pro víc PHP aplikací (patří různým uživatelům).
- Sdílené prostředí jednoho webového serveru.
- Všechny soubory musí být čitelné pro web server...
- To znamená: **všechny soubory jsou čitelné pro všechny uživatele**.

# Shared Hosting: řešení v PHP

- Řešení tohoto problému na úrovni PHP je architekturálně nekorektní, ale často používané.
- PHP řeší tento problém dvěma konfiguračními direktivami:
  - `open_basedir` – omezuje přístup k souborům na určité adresáře.
    - Relativně efektivní.
    - Nekomplikované.
  - `safe_mode` – omezuje přístup k souborům na vlastníka běžícího skriptu.
    - Pomalý a složitý přístup.
    - Mnoho omezení pro PHP aplikace.

# Security Through Obscurity

- Další techniky zabezpečení:
  - Vypnout PHP identifikační hlavičky (ini direktiva)

    ```
    expose_php=off
    ```
  - Vypnout Apache identifikační hlavičky (ini direktiva)

    ```
    ServerSignature=off
    ```
  - Nezveřejňovat <?php phpinfo(); ?>

# Nejčastější bezpečnostní chyby webových aplikací

- Cross Site Scripting (XSS)
- Injection Flaws (SQL injection)
- Malicious File Execution
- Insecure Direct Object Reference
- Cross Site Request Forgery (CSRF)
- Information Leakage and Improper Error Handling
- Broken Authentication and Session Management
- Insecure Cryptographic Storage
- Insecure Communications
- Failure to Restrict URL Access

- Zdroj: OWASP – Top Ten 2007 (http://www.owasp.org/index.php/Top_10_2007)

# Zdroje

- Ilia Alshanetsky: Guide to PHP Security
  http://www.phparch.com/pgps
- Chris Shiflett: Essential PHP Security
  http://phpsecurity.org/
- **http://php.vrana.cz**

# Otázky???

**Děkuji za pozornost...**