15

Izrada bezbednih veb aplikacija

U ovom poglavlju nastavljamo razmatranje zaštite veb aplikacija i to tako što prelazimo na opsežniju temu zaštite cele veb aplikacije. Pošto svaki pojedinačni deo veb aplikacija moramo zaštititi od mogućih zloupotreba (nenamernih ili namernih), potrebno je da definišemo određene strategije koje će omogućiti da naše aplikacije uvek budu zaštićene.

U ovom poglavlju razmotrićemo sledeće ključne teme:

- Strategije zaštite
- Zaštita programskog koda
- Zaštita veb servera i PHP-a
- Zaštita servera baze podataka
- Zaštita mreže
- Planiranje za slučaj katastrofe.

Strategije zaštite

Jedna od najkorisnijih mogućnosti interneta – otvorenost i mogućnost pristupanja svih računara svima ostalima – pokazuje se i kao jedna od najvećih glavobolja s kojom vi, kao autor veb aplikacije, morate da se izborite. Uz toliki broj računara na svetu, verovatno je da će neki među njihovim korisnicima imati sve samo ne plemenite namere. Kad pomislimo na sve opasnosti koje vrebaju na internetu, može biti zastrašujuće razmišljati o tome da se u globalnu mrežu postavi aplikacija koja radi s potencijalno poverljivim podacima kao što su brojevi kreditnih kartica, podaci o bankovnim računima ili zdravstveni kartoni. Ali, firme moraju da posluju, a mi kao autori aplikacija moramo gledati dalje od zaštite samo onih delova naših aplikacija koji se direktno odnose na e-trgovinu i moramo razviti način za planiranje i realizovanje zaštite. Suština je to da nađemo ravnotežu između potrebe da se zaštitimo i potrebe da se poslovanje nesmetano obavlja i da aplikacija radi bez problema.

Počnite s pravim načinom razmišljanja

Zaštita aplikacije nije tek jedna od mogućnosti. Kada pišete veb aplikaciju i razmatrate listu mogućnosti koje želite da ugradite, zaštita nije nešto što „onako, za svaki slučaj“ stavljate na listu i određujete jednog programera da na tome radi nekoliko dana. To mora

neprekidno biti sastavni deo osnovne strukture aplikacije i posao koji nikad ne prestaje, čak ni pošto aplikaciju pustite u redovan rad i smanjite obim razvoja, ili ga sasvim prekinete.

Ako od samog početka razmišljamo o zaštiti i planiranju aplikacije, kao i o načinima na koje bi se sistem mogao zloupotrebiti ili na koje bi napadači mogli pokušati da provale u njega, programski kôd možemo projektovati tako da se smanji verovatnoća pojavljivanja tih problema. To će nas takođe poštedeti potrebe da kasnije sve ponovo projektujemo iz početka kada najzad počnemo da se bavimo problemom zaštite (kada ćemo gotovo sigurno prevideti više drugih potencijalnih problema).

Ravnoteža između zaštite i upotrebljivosti

Jedna od najvećih briga tokom projektovanja sistema jesu lozinke korisnika. Korisnici će često birati lozinke koje nije posebno teško pogoditi pomoću softvera, naročito ako upotrebljavaju uobičajene reči iz govornog jezika. Potreban nam je način da smanjimo rizik pogađanja korisnikove lozinke i provale u sistem na taj način.

Jedno od mogućih rešenja je zahtevati da korisnik prođe kroz četiri prijavna dijaloga, svaki s drugom lozinkom. Možemo takođe tražiti da korisnik menja te četiri lozinke barem jednom mesečno i obezbediti da nikad ne ponovi lozinku koju je već koristio. Mogli bismo zahtevati da lozinke budu veoma dugačke i da sadrže različite vrste znakova (na primer, velika slova, mala slova, znakove interpunkcije i brojeve). Tako bi sistem postao znatno zaštićeniji, a zlonamerni hakeri bi morali da posvete mnogo više vremena zaobilaženju postupka prijavljivanja i provale u sistem.

Nažalost, takav sistem bi bio toliko zaštićen da ga niko ne bi koristio – pre ili kasnije bi korisnik pomislio da nije vredno truda. Znači, podjednako je važno da brinemo o zaštiti, koliko i o njenom uticaju na upotrebljivost sistema. Lako upotrebljiv i slabo zaštićen sistem može se pokazati kao privlačan za korisnika, ali će posledice biti i veća verovatnoća pojave bezbednosnih problema i mogući prekidi poslovanja. Slično tome, sistem čija je zaštita toliko robusna da je sistem na granici upotrebljivosti, privući će mali broj korisnika i takođe će negativno uticati na poslovanje firme.

Kao projektanti veb aplikacija, moramo uvek tražiti načine da poboljšamo zaštitu, ali da pritom ne umanjimo preterano upotrebljivost sistema. Kao i u oblasti korisničkog interfejsa, budući da ne postoje stroga i univerzalna pravila koja bi trebalo slediti, moramo se osloniti na sopstvene procene, testove upotrebljivosti i probne grupe da bismo utvrdili kako korisnici reaguju na naše prototipove i projekte.

Nadgledanje zaštite

Kada završimo razvoj veb aplikacije i postavimo je na servere kako bi ljudi počeli da je koriste, posao još nije završen. Deo zaštite aplikacije je i nadgledanje sistema tokom rada, kroz pregledanje dnevnika i drugih datoteka da bismo videli koliko dobro sistem radi i kako se koristi. Samo pažljivim praćenjem rada sistema (ili pisanjem alatki koje će obavljati deo tog posla umesto nas), možemo utvrditi postoje li bezbednosni problemi i otkriti oblasti kojima moramo posvetiti vreme da bismo razvili bezbednija rešenja.

Zaštita je, nažalost, neprekidna bitka i u izvesnom hiperboličnom smislu, bitka u kojoj ne možemo nikad pobediti. Neprekidan oprez, poboljšavanje sistema i brzo reagovanje na svaki problem čine cenu koja se mora platiti da bi veb aplikacija glatko radila.

Osnovni pristup problemu

Da bismo došli do najpotpunijeg rešenja zaštite, uz razumnu količinu truda i vremena, u ovoj knjizi ćemo opisati dvodelni pristup problemu zaštite. Prvi deo se nadovezuje na ono o čemu je dosad već bilo reči: kako planirati zaštitu aplikacije i kako u nju ugraditi mogućnosti koje će očuvati tu zaštitu. Kada bismo bili takvi da na sve lepimo etikete, takav pristup bismo mogli nazvati *pristup odozgo nadole*.

Nasuprot tome, drugi deo naseg pristupa problemu zastite mogli bismo nazvati *pristup odozdo nagore*. O tome će biti reči u ovom poglavlju. U toj fazi razmotrićemo pojedinačne komponente aplikacije, kao što su server baze podataka, sam server i mreža u kojoj se on nalazi. Pokušaćemo da ne samo rad s tim komponetama bude zaštićen, nego i da njihovo instaliranje i podešavanje bude zaštićeno. Pošto se mnogi proizvodi instaliraju s podrazumevanim konfiguracijama koje ostaju otvorene za napade, korisno je da saznamo gde su „rupe“ i da ih zatvorimo.

Zaštita programskog koda

Da biste se zaista pobrinuli za zaštitu svog koda, morate razmišljati na nivou pojedinačnih komponenata, tj. ispitati svaku komponentu pojedinačno i razmisliti o tome kako poboljšati njenu zaštitu. U ovom odeljku počećemo razmatranje onoga što možemo uraditi da bismo zaštitili svoj programski kôd. Mada ne možemo da vam pokažemo sve što biste možda mogli uraditi kako biste se zaštitili od svih mogućih bezbednosnih pretnji (tim temama posvećene su brojne knjige), možemo barem da navedemo nekoliko opštih smernica i usmerimo vas i dobrom pravcu.

Filtriranje podataka koje korisnik šalje

Jedna od najvažnijih stvari koje možemo uraditi u svojim veb aplikacijama da bismo ih bolje zaštitili jeste da *filtriramo sve podatke koje korisnik šalje*.

Projektanti aplikacija moraju da filtriraju sve podatke koji dolaze iz spoljnih izvora. To ne znači da bi trebalo projektovati sistem s pretpostavkom da su svi naši korisnici prevaranti. Mi i dalje želimo da se oni osećaju dobrodošli i da ih podstičemo da koriste našu veb aplikaciju. Samo želimo da budemo spremni za svaki pokušaj zloupotrebe našeg sistema.

Ako obezbedimo efikasno filtriranje ulaznih podataka, znatno ćemo umanjiti broj spoljnih pretnji i poboljšati robusnost sistema. Čak i ukoliko smo sasvim ubeđeni da možemo verovati korisnicima, ne možemo biti sigurni da oni nemaju određenu vrstu špijunskog ili drugog softvera koji menja postojeće ili šalje nove zahteve našem serveru. Znači, nikada ne treba verovati korisniku.

Zbog važnosti filtriranja podataka koje dobijamo od spoljnih korisnika, trebalo bi da razmotrimo načine na koje se to može uraditi.

Proveravanje da li smo zaista dobili jednu od očekivanih vrednosti

U nekim slučajevima ponudićemo korisniku skup vrednosti između kojih može da bira, za podatke kao što su način dostave (običnom poštom, preporučeno, kao specijalan paket), poštanski broj ili grad itd. A sada zamislite da imamo sledeći jednostavan obrazac, prikazan u listingu 15.1:

Listing 15.1 **simple_form.html – jednostavan obrazac**

```
<!DOCTYPE html>
<html>
<head>
   <title>What be ye laddie?</title>
</head>
<body>
<h1>What be ye laddie?</h1>

<form action="submit_form.php" method="post">

<p>
<input type="radio" name="gender" id="gender_m" value="male" />
   <label for="gender_m">male</label><br/>

<input type="radio" name="gender" id="gender_f" value="female" />
   <label for="gender_f">female</label><br/>

<input type="radio" name="gender" id="gender_o" value="other" />
   <label for="gender_o">other</label><br/>
</p>

<button type="submit" name="submit">Submit Form</button>
</form>

</body>
</html>
```

Taj obrazac može biti nalik onome na slici 15.1. Na tom obrascu pretpostavili bismo da – kad god ispitamo vrednost promenljive `$_POST['gender']` na stranici *submit_form.php* – možemo dobiti samo jednu od tri moguće vrednosti: `'male'`, `'female'` ili `'other'` – što je potpuno pogrešno.

Slika 15.1 Jednostavan obrazac za unošenje pola osobe.

Kao što je već rečeno, veb radi pomoću HTTP-a, jednostavnog tekstualnog protokola. Navedeni obrazac bio bi poslat na naš server u obliku tekstualne poruke sa strukturom nalik sledećoj:

```
POST /submit_form.php HTTP/1.1
Host: www.yourdomain.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
gender=male
```

Međutim, ne postoji baš ništa što bi nekog sprečilo da uspostavi vezu s našim serverom i šalje putem obrasca koje god hoće vrednosti. Prema tome, neko može da nam pošalje i sledeće:

```
POST /submit_form.php HTTP/1.1
Host: www.yourdomain.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 22
gender=I+like+cookies.
```

Ako napišemo sledeći kôd:

```
<?php
echo "<h1>
      The user's gender is: ".$_POST['gender']. ".
      </h1>";
?>
```

mogli bismo kasnije dobiti zbunjujuće rezultate. Znatno bolja strategija je da proverimo da li je poslata vrednost jedna od očekivanih/dozvoljenih vrednosti, kao u listingu 15.2:

Listing 15.2 **submit_form.php – provera podataka unetih u obrazac**

```
<?php
switch ($_POST['gender']) {
   case 'male':
   case 'female':
   case 'other':

      echo "<h1>Congratulations!<br/>
           You are: ".$_POST['gender']. ".</h1>";
   break;

   default:

      echo "<h1><span style=\"color:  red;\">WARNING:</span><br/>
           Invalid input value specified.</h1>";
   break;
}
?>
```

Sada je programski kôd nešto složeniji – sadrži listu prihvatljivih unosa – ali barem možemo biti sigurni da uvek dobijamo ispravne vrednosti, a to će postati znatno važnije kada budemo obrađivali finansijske vrednosti, koje su svakako poverljivije od pôla korisnika. Pravilo glasi: ne možemo nikad poći od pretpostavke da će vrednost koju dobijemo preko obrasca biti unutar skupa očekivanih vrednosti – to moramo prvo proveriti.

Filtrirajte čak i osnovne vrednosti

Elementima na HTML obrascu nije pridružen tip podataka i većina njih se prosleđuju serveru u obliku znakovnih nizova (koji predstavljaju podatke kao što su datumi, vremena ili brojevi). To znači sledeće: ako imate numeričko polje, ne možete pretpostaviti, niti očekivati da je podatak u polju zaista broj. Čak i u okruženjima gde se posebno moćnim programskim kodom na klijentskoj strani može pokušati da se obezbedi unošenje vrednosti samo određene vrste, nema garancije da te vrednosti neće nikad biti direktno poslate serveru, kao što smo videli u prethodnom odeljku.

Jednostavan način da obezbedite da vrednost bude određenog tipa jeste da je preslikate ili pretvorite u taj tip i da zatim radite s rezultujućom vrednošću, na sledeći način:

```
$broj_nocenja = (int)$_POST['br_nocenja'];
if ($broj_nocenja == 0)
{
  echo "GREŠKA: Pogrešan broj noćenja!";
  exit;
}
```

Ako korisnik unosi datum u nekom lokalnom formatu, kao što je *mm/dd/gg* za korisnike iz SAD, pomoću PHP-ove funkcije `checkdate()` možemo napisati kôd koji proverava da li imamo ispravan datum. Ta funkcija preuzima vrednosti *meseca*, *dana* i *godine* (dvocifrena godina) i pokazuje da li njihova kombinacija čini ispravan datum:

```
$mmddyy = explode($_POST['datum_polaska'], '/');
if (count($mmddyy) != 3)
{
  echo "GREŠKA: Zadali ste neispravan datum!";
  exit;
}
// obrada godina kao sto su 02 ili 95
if ((int)$mmddyy[2] < 100)
{
  if ((int)$mmddyy[2] > 50)
    $mmddyy[2] = (int)$mmddyy[2] + 1900;
  else if ((int)$mmddyy[2] >= 0)
    $mmddyy[2] = (int)$mmddyy[2] + 2000;
  // inače je datum < 0, što će otkriti funkcija checkdate()
}
if  (!checkdate($mmddyy[0],  $mmddyy[1],  $mmddyy[2]))
{
  echo "GREŠKA: Zadali ste neispravan datum!";
  exit;
}
```

Ako ulazne podatke filtriramo i proveravamo njihovu ispravnost, ne samo što ćemo obezbediti otkrivanje očekivanih grešaka (npr. da li je datum početka leta na avionskoj karti ispravan datum), što ionako moramo obezbediti, nego time možemo i poboljšati zaštitu celog sistema.

Priprema bezbednih tekstova SQL upita

Još jedan slučaj kada treba da obradimo znakovne nizove kako bismo ih učinili bezbednim jeste sprečavanje napada koji nastaju umetanjem SQL naredbi (engl. *SQL injection attacks*) i koje smo pomenuli kada smo razmatrali upotrebu MySQL-a u PHP-u. U tim napadima, zlonamerni korisnik pokušava da iskoristi propuste u loše zaštićenom programskom kodu i ovlašćenjima korisnika da bi izvršio dodatan SQL kôd. Ako ne obratimo pažnju, ime korisnika kao što je sledeće

```
mala_maca; DELETE FROM kupci;
```

može nam prouzrokovati priličan problem.

Tu vrstu narušavanja zaštite možete sprečiti paralelnom primenom dve osnovne metode:

- Koristite parametrizovane upite kad god je moguće. U takvim upitima, SQL kôd je razdvojen od podataka. Doduše, to neće pomoći za imena kolona i tabela, jer se ona ne mogu proslediti pomoću parametrizovanog upita. Međutim, pošto unapred znate svoju šemu, možete primeniti metodu bele liste sa odgovarajućim vrednostima.
- Proverite da li su svi ulazni podaci u skladu sa onim što očekujete da budu. Ako je pravilo da se imena korisnika sastoje od najviše 50 znakova i sadrže isključivo slova i brojeve, možemo biti sigurni da `"; DELETE FROM kupci"` na kraju imena verovatno nije nešto što bismo dozvolili. Ukoliko napišemo PHP kôd kojim se obezbeđuje da su ulazni podaci u skladu sa ispravnim i dozvoljenim vrednostima, pre nego što ih pošaljemo serveru baze podataka, znači da ćemo moći da dobijemo znatno razumljiviji opis greške nego što bi nam baza podataka prikazala (kada bi i proveravala tu vrstu stvari) i smanjićemo rizike.

Proširenje *mysqli* pruža dodatnu bezbednosnu prednost jer se dozvoljava izvršavanje samo po jednog upita pomoću metoda `mysqli_query` ili `mysqli::query`. Da biste izvršili više upita, morate upotrebiti metodu `mysqli_multi_query` ili `mysqli::multi_query`, čime se sprečava izvršavanje dodatnih, potencijalno štetnih naredbi ili upita.

Priprema izlaznih podataka

Gotovo podjednako važno kao filtriranje ulaznih podataka jeste *pripremanje izlaznih podataka*. Pošto u sistemu pripremimo podatke namenjene korisniku, od ključne je važnosti da oni budu takvi da ne mogu načiniti nikakvu štetu, niti biti uzrok neželjenih posledica. To postižemo pomoću nekoliko ključnih funkcija kojima se obezbeđuje da klijentski čitač tumači podatke koje mu šaljemo isključivo kao tekst koji treba da prikaže, a ne kao programski kôd koji treba da izvrši.

Mnoge veb aplikacije preuzimaju podatke koje je korisnik uneo a zatim ih prikazuju na određenoj veb stranici. Odlični primeri toga su stranice na kojima korisnik može da dodaje komentare na objavljen tekst, ili stranice sistema za razmenu poruka između korisnika (forumi). U takvim slučajevima moramo sprečiti korisnike da umeću zlonameran HMTL kôd u tekst.

Jedan od najlakših načina da to postignemo jeste pomoću funkcija `htmlspecialchars` ili `htmlentities`. Te funkcije preuzimaju određene znakove u ulaznom tekstu i pretvaraju ih u HTML *entitete*. Ukratko, HTML entitet je specijalan niz znakova, koji počinje znakom

ampersend (&), a svrha mu je da predstavi određeni specijalan znak koji se ne može lako predstaviti u HTML kodu. Iza znaka ampersend sledi ime entiteta i završni znak tačka i zarez (;). Entitet može biti i u obliku ASCII koda tastera, zadatog kao znak # kojem sledi broj, kao što je / što predstavlja kosu crtu (/).

Na primer, budući da su svi HTML elementi omeđeni znakovima < i >, može biti problem kako te znakove predstaviti u izlaznom sadržaju (jer će ih čitač veba standardno tumačiti kao graničnike HTML elemenata, a ne kao tekst). Da bismo rešili taj problem, koristimo entitete < i >. Slično tome, ako u HTML tekstu želimo da prikažemo znak ampersand, upotrebićemo entitet &. Polunavodnike i navodnike predstavlja entitet ', odnosno ". HTML klijent (čitač veba) pretvara entitete u običan tekst koji prikazuje i ne tumači ih kao delove HTML elemenata.

Razlika između funkcija `htmlspecialchars` i `htmlentities` jeste sledeća: prva funkcija zamenjuje samo znakove &, < i >, a mogu joj se zadati i opcioni parametri za polunavodnike i navodnike. Nasuprot tome, druga funkcija zamenjuje odgovarajućim entitetom sve što se može predstaviti tim imenovanim entitetom. Primeri takvih entiteta su simbol za autorska prava ©, koji se predstavlja pomoću entiteta © i simbol za novčanu jedinicu evro, koji se predstavlja kao €. Međutim, ta funkcija ne pretvara znakove u numeričke entitete.

Drugi parametar obe funkcije je vrednost koja određuje kako se tretiraju polunavodnici/navodnici i neispravne sekvence koda, a treći – opcioni – parametar obe funkcije jeste skup znakova u koji se kodira ulazni tekst (što je od ključne važnosti za nas jer želimo da ta funkcija ispravno radi s našim znakovnim nizovima u formatu UTF-8). Drugi parametar najčešće ima jednu od sledećih šest vrednosti:

- `ENT_COMPAT` (podrazumevana vrednost) – Navodnici se zamenjuju entitetom " ali ne i polunavodnici.
- `ENT_QUOTES` – Zamenjuju se i polunavodnici i navodnici, entitetima ' odnosno ".
- `ENT_NOQUOTES` – Funkcija ne zamenjuje ni polunavodnike ni navodnike.
- `ENT_IGNORE` – Neispravne sekvence koda se tiho zanemaruju.
- `ENT_SUBSTITUTE` – Neispravne sekvence koda zamenjuju se nekim Unicode zamenskim znakom (Unicode Replacement Character) umesto da se vrati prazan znakovni niz.
- `ENT_DISALLOWED` – Neispravne sekvence koda zamenjuju se nekim Unicode zamenskim znakom umesto da se ostave takve kakve su.

Pogledajte sledeći odlomak koda:

```
$input_str  =  "<p  align=\"center\">Korisnik je platio  \"15000?\".</p>
               <script type=\"text/javascript\">
               // ovde dolazi zlonameran JavaScript kôd.
               </script>";
```

Ako ga izvršimo u sledećem PHP skriptu (izlazni tekst prosleđujemo funkciji `nl2br` kako bismo bili sigurni da će ga klijentski čitač lepo formatirati):

```
<?php
  $str = htmlspecialchars($input_str, ENT_NOQUOTES, "UTF-8");
  echo nl2br($str);
  $str = htmlentities($input_str, ENT_QUOTES, "UTF-8");
  echo nl2br($str);
?>
```

dobili bismo sledeći tekst:

```
&lt;p align="center"&gt;Korisnik je platio "15000?".&lt;/p&gt;<br />
<br />
&lt;script type="text/javascript"&gt;<br />
// ovde dolazi zlonameran JavaScript kôd.
&lt;/script&gt;&lt;p align=&quot;center&quot;&gt;Korisnik je platio
&quot;15000&euro;&quot;.&lt;/p&gt;<br />
<br />
&lt;script type=&quot;text/javascript&quot;&gt;<br />
// ovde dolazi zlonameran JavaScript kôd.
&lt;/script&gt;
```

U prozoru čitača, tekst bi izgledao ovako:

```
<p align="center">Korisnik je platio "15000?".</p>

<script type="text/javascript">
// ovde dolazi zlonameran JavaScript kôd.
</script><p align="center">Korisnik je platio "15000?".</p>

<script type="text/javascript">
// ovde dolazi zlonameran JavaScript kôd.
</script>
```

Obratite pažnju na to da je funkcija `htmlentities` zamenila simbol za novčanu jedinicu evro (€) entitetom (`&euro;`), dok funkcija `htmlspecialchars` to nije učinila.

U slučajevima kada korisnicima dozvoljavamo da unose i HTML kôd, kao na forumima za razmenu poruka, gde bi ljudi možda voleli da koriste znakove koji definišu font, boju i stil teksta (podebljana ili kurzivna slova), moraćemo da ispitujemo ulazne znakovne nizove kako bismo otkrili te znakove i ostavili ih u tekstu.

Organizovanje programskog koda

Dobar osnovni pristup je sledeći: nijedna datoteka koja nije direktno dostupna korisniku preko interneta, ne bi trebalo da se nalazi u stablu direktorijuma u kome su smešteni dokumenti veb aplikacije. Na primer, ako je `/home/httpd/messageboard/www` korenski direktorijum nase veb lokacije za razmenu poruka, sve datoteke za umetanje trebalo bi da stavimo na neko mesto poput `/home/httpd/messageboard/lib`. Zatim, kada nam u programskom kodu zatreba da umetnemo sadržaj iz tih datoteka, možemo napisati:

```
require_once('../lib/user_object.php);
```

Za tu meru opreza postoji nekoliko razloga.

Prvo, razmotrite šta se događa kada zlonamerni korisnik pošalje zahtev za datoteku koja nije `.php` ili `.html`. Mnogi veb serveri – ukoliko nisu ispravno konfigurisani – samo će proslediti ceo sadržaj tražene datoteke u izlazni tok. Ako datoteku `neka_biblioteka.inc` čuvate negde u stablu direktorijuma za dokumente veb lokacije, može se dogoditi da korisnik vidi ceo vaš programski kôd u svom čitaču veba. Tako bi korisnik mogao da vidi podatke ili putanje servera i da otkrije bezbednosne propuste koje ste možda prevideli.

Da bismo to sprečili, moramo podesiti veb server tako da prihvata samo zahteve u kojima se traže `.php` i `.html` datoteke, dok za druge vrste datoteka (kao što su `*.inc`, `*.mo`, `*.txt` itd.) treba da prijavljuje grešku.

Drugo, čak i ako sve vaše datoteke imaju oznaku tipa `.php`, neke od njih koje su namenjena za umetanje u kôd pomoću komande `include` mogu izazvati nepoželjne posledice ukoliko se učitaju van konteksta. Razmotrite, na primer, biblioteku koda za administriranje. Možete tražiti proveru identiteta u uobičajenom kontekstu, ali ako se datoteka učita sama – ta provera se može narušiti.

Slično tome, sve ostale datoteke – kao što su one s lozinkama, tekstualne datoteke, konfiguracione datoteke ili specijalni direktorijumi – moraju se čuvati izvan stabla direktorijuma javno dostupnih dokumenata. Čak i ako mislite da je veb server ispravno podešen, možda ste nešto prevideli; ili, ako kasnije premestite veb aplikaciju na nov server koji nije pravilno konfigurisan, može se dogoditi da otvorite mogućnost provale.

Ukoliko u datoteci `php.ini` uključite opciju `allow_url_fopen` – a imajte na umu da je ona podrazumevano uključena – teorijski biste mogli da umećete ili zahevate datoteke sa udaljenih servera. Pošto bi to bila još jedna mogućnost narušavanja zaštite, trebalo bi da izbegavate izvršavanje datoteka koje se nalaze na udaljenim mašinama, naročito na onim nad kojima nemate potpunu kontrolu. Slično tome, odluke o tome koje ćete datoteke umetati ili zahtevati, ne bi trebalo da zasnivate na podacima koje je korisnik poslao jer i u tom slučaju zlonamerni ili pogrešni podaci mogu biti uzrok problema.

Šta treba da se nalazi u programskom kodu

Mnogi blokovi koda za pristupanje bazama podataka koje ste dosad videli sadrže u samom kodu ime baze podataka, korisničko ime i lozinku, sve u obliku otvorenog teksta, kao u sledećem primeru:

```
$conn = @new mysqli("localhost", "mika", "tajna", "nekabaza");
```

Mada je to zgodno i jednostavno, prilično je nebezbedno jer ako se zlonamernici domognu ove datoteke, dobiće širom otvoren pristup bazi podataka, sa svim ovlašćenjima koje ima korisnik `mika`.

Znatno je bolje da korisničko ime i lozinku čuvate u datoteci koja se ne nalazi u stablu direktorijuma za dokumente veb aplikacije i da je umećete u skript kad vam zatreba, na sledeći način:

```
<?php
  // ovo je dbconnect.php
  $db_server = 'localhost';
  $db_korisnik = 'mika';
  $db_lozinka = 'tajna';
  $db_ime = 'nekabaza';
?>
```

Ova datoteka se onda može pozvati ovako:

```
<?php
  include('../code/dbconnect.php');
  $conn = @new mysqli($db_server, $db_korisnik, $db_lozinka,
                      $db_ime);
  // itd.
?>
```

Preporučljivo je da uradite isto i za sve ostale slične poverljive podatke za koje bi vam trebao dodatan sloj zaštite.