**Bezbednosni koncepti**

Jedinica: 15 od 19

**Rezime**

* Bezbednost PHP web aplikacije najranjivija je na ulazu;
* Korisnički ulaz vrši se putem korisničkih kontrola;
* Svaki nesiguran podatak koji ulazi u PHP aplikaciju treba filtrirati;
* U procesu filtracije razlikujemo pojmove crne i bele liste;
* Na ulazu treba proveravati kontrole i referera;
* Na izlazu treba proveravati korisnički definisane sadržaje;
* Kod baze podataka treba proveravati SQL upite;
* Register globals omogućava automatsko kreiranje promenljivih po nazivu parametra;
* Da bi aplikacija bila bezbednija, register globals treba da bude isključen;
* Prilikom uključenja postojećeg koda, treba vršiti proveru uključenih fajlova, najbolje putem bele liste.

U ovoj lekciji upoznaćemo se sa bezbednosnim konceptima na vebu. Web sajtovi se, kao što znamo, sastoje od klijentskih i serverskih komponenti. Pri toj klasifikaciji, možemo reći da, sve dok se naš sajt nalazi u kontekstu klijenta, sistem je potpuno bezbedan, jer zapravo i nema nikakve veze sa serverom. Ali, u trenutku kada se odvija neki korisnički definisan proces na serveru, jedan sajt postaje ranjiv.

Pogledajmo šta se događa u procesu emitovanja jedne statičke HTML strane. Klijentska aplikacija šalje zahtev web serveru i web server odgovara tako što pronalazi i emituje traženi HTML dokument. U ovakvom procesu nema prostora ni za šta, osim za pomenuti niz aktivnosti, pa je zbog toga ovakva aplikacija praktično neuništiva.

U procesu kreiranja i emitovanja dinamičke web strane ovaj proces ima još nekoliko koraka. Na početku, i ovde klijentska aplikacija traži određeni dokument, ali server, umesto da pronađe i prosledi taj dokument, prosleđuje serverskoj skripti ceo zahtev i ona ga onda obrađuje i emituje na izlazu (klijentu). Ova obrada je ključna tačka za bezbednost jedne web aplikacije, jer ukoliko korisnik uspe da infiltrira svoj deo koda u serversku skriptu, imaće neograničene mogućnosti za manipulaciju serverom.

Dakle, aplikacija je najranjivija na ulazu. Zbog toga je i najbitnije da budemo sigurni u to šta u nju ulazi; a tu sigurnost ćemo postići tako što ćemo kontrolisati sve ulaze.

Šta su uopšte ulazi u web aplikaciju?

Da bi korisnik uspeo da dođe do serverskog koda neke aplikacije (kroz tu aplikaciju), potrebno je da joj se obrati putem nekih parametara. Ovi parametri obično dolaze do aplikacije kroz forme (post) ili parametrizovani URL string (get). Kada neki od ovih parametara dođe do servera, server ga smešta u odgovarajuću promenljivu. Ove promenljive su jedinstvene i dostupne su kompletnom kontekstu aplikacije, te se zato nazivaju superglobalne promenljive.

Većina superglobala, prilikom svakog zahteva i odgovora, prelazi put od klijenta do servera i obrnuto, pa se zato smatraju nečistima, i potreban im je poseban tretman kako bi njihova upotreba bila zagarantovano bezbedna.

Pošto u ovom trenutku ne znamo sve što nam je potrebno da bismo obradili sve što se tiče bezbednosti u PHP-u (jer, zapravo, još uvek nismo ni počeli da obrađujemo web aplikacije), na bezbednost ćemo se vraćati više puta kroz kurseve.

**Pojam crne i bele liste**

Već smo rekli da se dosta ulaza u samu aplikaciju događa putem kontrola koje su generalno nesiguran izvor. Zato se, prilikom svakog takvog ulaza informacija, vrši određena filtracija. Pre nego što počnemo da opisujemo načine te filtracije, treba da osmislimo šta ćemo filtrirati.

Kada filtriramo podatke, možemo reći aplikaciji jednu od dve stvari:

* + ne puštaj nikoga ko nema određene uslove,
  + puštaj samo one koji imaju određene uslove.

Ova dva koncepta nazivaju se crna i bela lista.

Razlika između ovih koncepata je u tome što crna lista zahteva mnogo manje pažnje, jer, nakon što navedemo objekte koji nemaju pristup strukturi, ona će biti dostupna svim ostalim objektima.

Bela lista se smatra boljim bezbednosnim konceptom od crne liste, jer je unos ograničen samo na očekivane vrednosti, pa tako neželjeni objekat ima mnogo manje šanse da prođe.

**Ulaz**

Prva ranjiva tačka u sistemu je, rekli smo, superglobalna promenljiva. A to je ujedno i mesto gde naš serverski kod ima mogućnost neke kontrole.

Prvo što možemo prekontrolisati jeste da li nam je korisnik validan ili ne. U slučaju da se radi o korisniku koji nije registrovan u našem sistemu, možemo proveriti odakle je došao (ukoliko treba da obradimo podatke iz neke forme, ta forma mora doći sa nekog sigurnog i poznatog izvora). Lokacija sa koje je korisnik došao na našu stranu naziva se **referer**.

Ukoliko očekujemo da korisnik bude registrovan u sistemu, vršimo sistemsku proveru  (putem cookieja, sessiona ili baze).

Ovo je jedan od najbezbednijih načina prosleđivanja podataka.

Kada je korisnik proveren, sledeća tačka je sam unos. Odnosno, superglobali koji taj unos sadrže. U web aplikacijama, korisnik može ozbiljno da nam naudi jedino putem serverskog ili SQL skripta. Zato je takav unos najčešće potrebno preduprediti; najbolje tako što ćemo ograničiti korisnika na unos isključivo validnog sadržaja.

Na primer, ako naša aplikacija poseduje jednu kontrolu za unos teksta, i u tu kontrolu treba uneti ime ili prezime, mi možemo na serveru proveriti da li je korisnik uneo isključivo slova. Ukoliko tekst sadrži specijalni karakter ili broj, možemo eliminisati podatak i zatražiti novi unos.

|  |  |
| --- | --- |
| 1 | ctype\_alpha("myname"); |

Ova funkcija vratiće rezultat *true*, ukoliko parametar sadrži samo alfabet karaktere ili ukoliko korisnik treba da unese neku brojnu vrednost:

|  |  |
| --- | --- |
| 1 | is\_numeric("1234"); |

Funkcija vraća *true* ukoliko je uneseni parametar broj.

**Izlaz**

U PHP-u verovatno će izlaz naše aplikacije biti sam HTML dokument ili eventualno neka serijalizovana struktura podataka. U bilo kom slučaju, trenutak u kome će naša aplikacija prikazati svoje nedostatke upravo je izlaz.

Na primer, ukoliko je korisnik uneo klijentsku Cross Site skriptu (o kojima će biti više reči u narednoj lekciji), ona će biti manifestovana baš na izlazu, dok na sam sistem neće imati veliki uticaj. S druge strane, klijentska skripta može naneti posledice klijentu.

Zato je potrebno izvršiti neku vrstu filtracije i na samom izlazu. Ova filtracija, pre svega, podrazumeva uklanjanje svega što može da bude izvršeno u vidu skripte, što bi bilo jednostavno kada ponekad na strani ne bi trebalo da bude emitovan baš sadržaj neke takve skripte. Na primer, forum koji se bavi klijentskim skriptiranjem, pa svi na taj forum šalju kodove svojih skripti. Ako bismo napravili takvu filtraciju koja bi eliminisala sve skripte, postovi na ovom forumu ne bi imali svrhu. Zato se u ovakvim situacijama vrši takozvani HTML enkoding (i o ovome će biti više reči u sledećoj lekciji), koji emituje HTML sadržaje (klijentske skripte i sl.), ali u onesposobljenoj formi, tako što će njihove tagove emitovati kroz njihovu HTML enkodiranu reprezentaciju. Ovo funkcioniše jednostavnije nego što zvuči. Znamo da se skripte nalaze u tagovima <script>. Kada pretraživač naiđe na ovakav tag, on emituje klijentsku skriptu. Ali, ako naiđe na sledeći tag: &lt;**script**&gt; emitovaće tekst: *<script> ,* a da taj tekst ne tretira kao početak skripte.

Naravno, za ovako nešto možete kreirati sopstvenu funkcionalnost koja će obrađivati stringove na način koji vam odgovara ili koristiti neke već gotove funkcije:

* + htmlspecialchars("<script>");

Pretvoriće zadati string u &lt;script&gt;

* + htmlspecialchars\_decode("&lt;script&gt;");

i obrnuto... Pretvoriće HTML enkodirani string u običan.

Ali, često vam gotova rešenja neće odgovarati (jer, na primer, nešto ipak želite da propustite). Konkretno, funkcija iz primera ne enkodira jednostruke navodnike koji umeju da nanesu probleme prilikom rukovanja sa bazom (mada to možete uvesti kao parametar (*htmlspecialchars("'''", ENT\_QUOTES)*)).

**Register globals**

Ovo može biti bezbednosno veoma slaba tačka web aplikacije. Register globals je opcija koja se nalazi u php.ini fajlu, ali je moguće podesiti i kroz **ini\_set()** funkciju (koja podešava parametre kroz samu aplikaciju).  
  
Ovo je moguće preduprediti i ukoliko na početku koda postavite inicijalnu vrednost promenljivoj, ali je svakako najbolje opciju register globals deaktivirati.

Naznaka register globalsa je ovde spomenuta kao napomena za korisnike koji rade sa starijim verzijama PHP-a. Od verzije PHP 5.4.0 ova funkcionalnost se smatra prevaziđenom i uklonjena je iz redovne postavke PHP-a.

**Baza podataka**

Još jedna kritična bezbednosna tačka jedne PHP aplikacije je i izvor podataka (govorimo o ozbiljnijoj aplikaciji koja podrazumeva i izvor podataka). Sami izvori mogu biti baza, fajlovi ili mreža.

Kada je baza u pitanju, njena najranjivija tačka su upiti koje joj upućujemo, pa je zato naročito bitno dobro isfiltrirati korisničke podatke pre nego što ih prosledimo bazi, da ne bi došlo do SQL injectiona (takođe, nešto više reči i o ovome u sledećoj lekciji). Za filtraciju možemo koristiti već pomenute ili sopstvene funkcije.

Često se za sadržaj koji služi za proveru (šifre) koristi i neki jednosmerni algoritam za enkodiranje. Ovo je odlična bezbednosna praksa, jer, nakon što je šifra jednom enkodirana na ovaj način, više se ne može doći do nje (osim ako je poznata).

|  |  |
| --- | --- |
| 1 | $sifra=md5("mojaSifra"); |

Ovako enkodiranu šifru ne možete više povratiti, već samo proveriti, i to tako što ćete je za proveru ponovo enkodirati:

$sifra=md5("mojaSifra");  
if($sifra=="ab6890bf5bb25b645a5c671d9ff61a3a")  
.... neka bezbednosna logika aplikacije

Vaš kod neće, naravno, nikada izgledati ovako (jer je ovde šifra hardkodirana u program), već će kriptovani deo biti izvlačen iz nekog izvora (baze) i na taj način poređen.

**Fajl sistem**

Kao što znamo, PHP može direktno pristupati fajl sistemu, što može omogućiti potencijalnom napadaču razne neželjene intervencije na našoj aplikaciji. Ono što je najopasnije jesu trenuci kada uključujemo fajlove u aplikaciju (include, require itd.). Ako, na primer, naša aplikacija funkcioniše tako što, na osnovu parametra, učitava određeni fajl, korisnik može lako kroz taj parametar proslediti sopstveni fajl (na sopstvenom serveru), koji će, iako će biti emitovan sa potpuno druge lokacije, funkcionisati u kontekstu našeg servera i aplikacije i tako imati sva prava kao i bilo koja naša aplikacija.

Na primer, ako naša aplikacija gradi stranu, tako što na osnovu parametra učitava određeni fajl:

include $GET\_['strana'] + ".php"; // $GET je asocijativni niz parametara iz url stringa

Tada bi korisnik mogao, jednostavno, umesto našeg URL stringa (koji je npr. ?strana*=*mojaStrana)*http://korisnikovdomen.com/korisnikovFajl* (pri čemu na tom serveru stvarno postoji fajl: korisnikovFajl.php).

Tada bi korisnikov Fajl.php bio implementiran u našu stranu, a njegov sadržaj bi, radeći u konetekstu naše aplikacije, mogao biti toliko loš po nas, kolika bi bila i mašta našeg napadača.

Kao i za prethodne vrste napada, i za ovaj postoje jednostavna rešenja, koja se baziraju na crnim ili belim listama. U ovom slučaju možemo napraviti belu listu dozvoljenih fajlova i proveravati je prilikom svakog učitavanja eksternog fajla. Na primer:

$dozvoljeneStrane = array("mojaStrana","tvojaStrana");  
if in\_array($\_GET['strana'],$dozvoljeneStrane);  
include $strana + ".php";

U ovoj lekciji smo se dotakli nekih bezbednosnih koncepata, koji se tiču PHP-a i njegovog okruženja. Za sada je dovoljno da znate samo kako bezbednost u PHP-u funkcioniše, kao i najosnovnije načine primene bezbednosnih mehanizama, a kasnije, kroz ostale kurseve i lekcije obrađivaćemo usputno bezbednosne tehnologije kroz trenutno obrađivanu tehnologiju.

Ovo su standardni bezbednosni koncepti PHP-a i pre se mogu svrstati u savladavanje dečjih bolesti, nego prave opasnosti (ne zato što nema opasnosti, već zato što smo te opasnosti svesni i znamo kako da je uklonimo), pa ih tako i tretirajte – kao vakcine za vaše aplikacije protiv poznatih bolesti.

**Kako možemo opisati pojam bele liste?**

 puštaj samo one koji ispunjavaju određene uslove

 zabrani pristup onima koji zadovoljavaju uslove

 ne puštaj nikoga

 puštaj sve

**Vežba 1**

***Problem:***

U aplikaciju ulazi promenljiva:

$strana = "allUsers.php";

Potrebno je napraviti mehanizam koji će putem bele liste obezbediti bezbedno učitavanje ove strane sa sistema.

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <?php  $allPages = array("allUsers.php","anyOtherPage.php");  $page = "allUsers.php";  if(in\_array($page,$allPages))          include $page;      else          echo "unknown page";  ?> |

Ideja validacije putem bele liste podrazumeva postojanje liste unosa koje smatramo dozvoljenim. U našem slučaju je to niz dozvoljenih fajlova za inkludovanje:

|  |  |
| --- | --- |
| 1 | $allPages = array("allUsers.php","anyOtherPage.php"); |

Promenljiva $page sadrži ulazni podatak koji treba proveriti u listi. Za ovaku proveru koristimo if kontrolu toka u kombinaciji sa funkcijom in\_array. Ova funkcija proverava da li se traženi element nalazi u nizu. Ukoliko rezultat pretrage bude pozitivan, na mesto poziva se vraća boolean vrednost TRUE, u suprotnom FALSE. Ukoliko unos odgovara listi, vrši se inkludovanje:

|  |  |
| --- | --- |
| 1 | include $page; |

u suprotnom se prikazuje obaveštenje:

|  |  |
| --- | --- |
| 1 | echo "unknown page"; |

**Vežba 2**

***Problem:***

Potrebno je napraviti validaciju sesije na osnovu User Agenta. Nakon uspešne validacije, potrebno je regenerisati ID sesije. Ukoliko validacija nije dobra, potrebno je prekinuti izvršenje aplikacije i emitovati poruku da korisnik nije validan.

***Rešenje:***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?php      if(!isset($\_SESSION))          session\_start();        if(!isset($\_SESSION['ua']))          die("Invalid user");        $currentUserAgent = $\_SERVER['HTTP\_USER\_AGENT'];      $sessionUserAgent = $\_SESSION['ua'];        if($sessionUserAgent != $currentUserAgent)          die("Invalid user");      echo "valid user";      session\_regenerate\_id(true);  ?> |

Proveravamo sesiju i ukoliko je potrebno vršimo njeno startovanje. Takođe, proveravamo da li je setovan ključ „ua" i, ukoliko nije, prekidamo dalje izvršavanje programa. Promenljiva $currentUserAgent dobija trenutnu vrednost za $\_SERVER['HTTP\_USER\_AGENT'], dok promenljiva $sessionUserAgent dobija vrednost za $\_SESSION['ua'], odnosno poznatu vrednost za HTTP\_USER\_AGENT.  
  
Sada možemo proveriti da li se vrednosti ove dve promenljive ne poklapaju. Ukoliko je to slučaj, potrebno je prekinuti izvršavanje programa. Ukoliko ispunjavanje ovog uslova nije slučaj, vrši se prikaz poruke obaveštenja i generisanje novog id-ja sesije.

|  |
| --- |
| **Napomena**    U primeru se pominje superglobalna promenljiva $\_SESSION. U njoj se obično skladište podaci o aktivnoj sesiji korisnika. O sesijama će biti više reči u narednim kursevima i za sada je možete tretirati samo kao najobičniji asocijativni niz. Funkcija session\_start pokreće sesiju, a funkcija session\_regenerate\_id ponovo generiše broj sesije. |